Universidad de las Ciencias Informáticas
Facultad 2

Título: Defensa preventiva contra programas malignos.

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor(es): Aimé Rodríguez Begdadi

Tutor(es): Ing. Yohamnes Hernández Báez

Marzo, 2007

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 2 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Aimé Rodríguez Begdadi
_____

Yohamnes Hernández Báez
_____

DEDICATORIA

A mis padres, por darme su apoyo y la actitud ante la vida que me hizo llegar a este momento; a mi hermana, inseparable compañera en la aventura de conocer el mundo; al amor de mi vida, por llevarme a lugares dentro de mi misma que ignoraba; al fruto de ese amor que ahora crece en mí.

RESUMEN

Esta investigación aborda el estudio de los programas malignos, sus clasificaciones así como las acciones más comunes que realizan. Se determina cuales de estas acciones resultan factibles de interceptar, se proponen métodos para detectarlas luego de realizar un estudio del proceso de llamada a los servicios del sistema. Se realiza un estudio de las posibles vulnerabilidades con las que un programador se podría encontrar a la hora de implementar los métodos propuestos. Para culminar se desarrolla una prueba de concepto con el objetivo de demostrar la validez y efectividad de los métodos propuestos para la intercepción. Esta investigación y su prueba de concepto servirán como base para la empresa de software SEGURMATICA para el desarrollo de aplicaciones de seguridad basadas en la defensa preventiva.

PALABRAS CLAVE

Programas malignos, intercepción, acciones malignas, hook.

TABLA DE CONTENIDOS

INTRODUCCIÓN.

Corría el año 1980 y ya el primer virus, conocido como “Elk Cloner”, había salido a la luz. Como antecedentes, se pueden encontrar numerosas investigaciones sobre inteligencia y vida artificial, llegando a resaltar John Von Neuman como el primero en manejar estos conceptos quien a comienzos de la década de 1950 establece por primera vez la idea de programas auto replicables. Más tarde, en 1960 en los laboratorios de Bell se comienzan a desarrollar juegos (programas) que "luchan" entre sí con el objetivo de lograr el mayor espacio de memoria posible. Estos programas llamados “Core Wars” hacían uso de técnicas de ataque, defensa, ocultamiento y reproducción que son sin duda alguna, el preámbulo de lo que hoy en día son conocidos como programas malignos.

Ya a mediados de los noventa se produjeron enormes cambios en el mundo de la informática personal que llegan hasta nuestros días y que dispararon el número de programas malignos en circulación hasta límites insospechados. Si a finales de 1994 el número de estos, según la International Computer Security Association (ICSA), rondaba los cuatro mil, en los siguientes cinco años esa cifra se multiplicó por diez, y promete seguir aumentando.

Cientos de programas malignos son descubiertos mes a mes (de 6 a 20 por día), y técnicas más complejas se desarrollan a una velocidad impresionante a medida que el avance tecnológico permite la creación de nuevas puertas de entrada. A la par del desarrollo de los programas malignos y para dar solución a esta problemática, se crean programas antivirus que protegen las computadoras de la mayoría de los ataques producidos.

Un antivirus es una gran base de datos con la huella digital de todos los programas malignos conocidos para identificarlos. Los fabricantes de antivirus avanzan tecnológicamente casi en la misma medida que lo hacen los creadores de virus. Esto sirve para combatirlos, aunque no para prevenir la infección o la creación de otros nuevos. Actualmente existen técnicas, conocidas como heurísticas, que brindan una forma de "adelantarse" a los nuevos programas malignos. Estas técnicas son sumamente útiles para las infecciones que todavía no han sido actualizadas en el antivirus porque tratan de localizar los programas malignos de acuerdo a ciertas características ya preestablecidas. De esta manera el antivirus es capaz de analizar archivos y documentos y detectar estructuras de ficheros sospechosas. Esta posibilidad puede

ser explotada gracias a que de los 6 a 20 nuevos programas malignos diarios, sólo aparecen unos cinco totalmente novedosos al año constituyendo estos cinco novedosos los que presentan el mayor peligro para los sistemas.

La principal ventaja de este método es que puede detectar tantos virus conocidos como desconocidos, ya que se centra en la búsqueda de características comunes a todos los virus. Sin embargo, el análisis heurístico tiene varios inconvenientes, muestra una mayor lentitud respecto a otras técnicas de análisis y poca precisión de los datos que aporta sobre los programas malignos detectados, generando un elevado número de falsos positivos.

Básicamente todos los sistemas antivirus tienen un funcionamiento similar, interceptan los accesos a sistemas de ficheros y realizan una búsqueda de patrones prefijados que han sido cargados por medio de las actualizaciones y mediante los cuales se identifican programas malignos determinados, pero a pesar de toda la robustez que pueda tener un antivirus en sus métodos de búsqueda, el punto más débil, radica en el hecho de que el sistema puede ser atacado por un programa maligno para el cual no se haya realizado aún una actualización. Además si analizamos la velocidad de propagación de los programas malignos, podremos darnos cuenta de que estamos ante un problema que no puede ser ignorado.

A raíz de lo antes expuesto la situación problémica que se vislumbra es que:

- Los antivirus no pueden detectar un programa maligno para el cual no se haya generado una actualización y muchas veces la utilización de técnicas heurísticas no resulta del todo eficiente.

Es por ello que surge el problema científico siguiente:

- ¿Cómo proteger los sistemas de los programas malignos, para los cuales aún el antivirus no ha generado actualización, por medio de métodos preventivos de defensa?

Se hace necesaria entonces la búsqueda de nuevas alternativas de detección que sean capaces de adelantarse en pensamiento a los programas malignos, es decir, una defensa anticipada, que permita

detectar las acciones más comunes llevadas a cabo por los programas malignos y evitar que estas se realicen.

Actualmente, el Kaspersky Anti-virus en su versión 6.0 incorpora en sus opciones lo que ellos denominan “proactive defense”. Durante el tiempo que la defensa preventiva esté habilitada, el antivirus, constantemente interceptará acciones sospechosas y se las comunicará al usuario. Para hacer menos molesta esta tarea el usuario tiene la opción de crear una regla de trabajo a fin de que la misma acción no sea interceptada dos veces, si es que en realidad es una acción deseada.

A pesar de que ya este moderno antivirus tiene adelantado bastante en términos de defensa preventiva, como es común en el mundo del desarrollo del software, toda esta información permanece oculta, siendo el único lugar para aprenderla los mismos laboratorios Kaspersky. No obstante como es una idea que se ha empezado a aprovechar solo en los últimos tiempos aún se encuentra a las orillas del conocimiento que se podría abarcar con ella.

Por todo lo antes mencionado el objeto de la investigación a desarrollar será el siguiente:

- Acciones que realizan los programas malignos y sus métodos de detección.

El campo de acción en que se enmarcarán las tareas de la investigación será:

- Las acciones más comunes que realizan los programas malignos y los métodos documentados por el sistema operativo, si existen, para detener o interceptar estas acciones.

El objetivo general de este trabajo radica en:

- Proponer métodos eficaces de prevención capaces de detectar y detener las acciones malignas más comunes.

Para lograr el mismo se proponen los siguientes objetivos específicos:

- Desarrollar una investigación sobre el comportamiento más generalizado de los programas malignos a fin de conocer las acciones principales que estos realizan.

- Realizar un estudio sobre la posibilidad real en el sistema operativo de la intercepción de las acciones más comunes llevadas a cabo por los programas malignos.

- Realizar un análisis de las vulnerabilidades que podrían hacer que los métodos de intercepción detectados cayeran en un completo fracaso.

- Realizar pruebas de concepto que validen los métodos de intercepción detectados.

El desarrollo de esta investigación será un aporte decisivo en el fortalecimiento del producto antivirus cubano desarrollado por la empresa SEGURMATICA, además de constituir un libro abierto al conocimiento de los programas malignos y las técnicas modernas de prevención.

Para lograr desarrollar la investigación, se proponen una serie de tareas a las cuales se les dará solución en cada uno de los capítulos. Se utilizarán métodos teóricos y dentro de estos los analítico-sintéticos.

A lo largo del primer capítulo, se estudiarán los conceptos y términos utilizados en el trabajo, así como los necesarios para la comprensión de la materia. Esto no constituye un objetivo de la investigación pero facilitará a los lectores la comprensión de la misma. Se analiza además el comportamiento general de los programas malignos para concluir en las acciones realizadas por estos que son realmente sospechosas y que serán el foco de atención en el resto del trabajo.

En el segundo capítulo se estudiarán las acciones malignas más comunes para con esta información determinar el mecanismo de detección e intercepción para cada una de ellas. Finalmente se obtendrán métodos que podrán ser utilizados en un sistema antivirus.

En el capítulo tercero se desarrollará un estudio de las vulnerabilidades que pueden presentar los métodos propuestos, y que afectarían el proceso de intercepción y en el peor de los casos la estabilidad del sistema.

El capítulo cuatro constituirá una prueba de concepto, donde se demostrará si las metodologías para interceptar las posibles acciones malignas cumplen verdaderamente con el objetivo deseado.

CAPÍTULO 1. PROGRAMAS MALIGNOS. CLASIFICACIÓN Y COMPORTAMIENTO.

Introducción.

En este capítulo se aborda el estudio de los programas malignos, sus clasificaciones, así como las acciones más comunes que realizan, permitiendo llegar a la conclusión de cuáles de ellas resultan verdaderamente sospechosas con poca probabilidad de falsos positivos y si son factibles de interceptar.

1.1 Programas malignos. Terminologías.

En la época actual existen muchas variantes de programas malignos que pueden afectar los sistemas y son nombrados de distintas maneras de acuerdo a sus técnicas de infección y a la acción que realizan. Aunque a veces estos términos son conocidos por los programadores no pudiéramos profundizar en el tema sin antes aclarar algunas terminologías usadas para denominar los programas malignos:

Virus: Se le llama así a un código ejecutable que es capaz de reproducirse recursivamente a través de ficheros o áreas del sistema. Generalmente se denomina virus a todo programa maligno, y aunque este término está bastante difundido, en algunos momentos es necesario aclarar de que tipo de programa maligno se habla en realidad pues solo se considera puramente un virus a aquel código que no tiene vida propia fuera del fichero hospedero.

Gusanos: Los gusanos son programas malignos independientes de ficheros hospederos y típicamente se transmiten a través de la red. Son programas autoreplicables que no alteran los archivos, sino que residen en la memoria y se duplican a ellos mismos.

- **Gusanos de Correo:** Son gusanos especiales capaces de enviarse a ellos mismos por correo.

- **Octopus:** Es un tipo sofisticado de gusanos que existen como un conjunto de programas en más de una computadora en una red.

- **Conejos:** Es un gusano que salta de un lugar a otro, existiendo solo una copia en un tiempo determinado.

Bombas lógicas: Son programas que tienen un funcionamiento durante un tiempo hasta que se cumplen las condiciones necesarias. En caso de que las condiciones no se cumplan, nada ocurre.

Caballos de Troya: Son programas que brindan funcionalidades a administradores remotos sin conocimiento del usuario de la computadora. Generalmente no causan daños sino que se dedican a reunir información necesaria para el atacante. Una variante son los llamados Backdoors, que escuchan por un puerto y permiten al atacante conectarse a la máquina, o sea abren huecos en la seguridad del sistema dejándolo vulnerable a actos maliciosos.

Generadores de Programas Malignos (Kits): Los escritores de programas malignos han desarrollado Kits con el objetivo de generar nuevos programas malignos de forma automática. Estos no son más que aplicaciones basadas en menús en los cuales se seleccionan las características del programa maligno a crear.

Rootkits: Un rootkit no es más que una herramienta, o un grupo de ellas usadas por intrusos con fines maliciosos para esconder los procesos y los archivos que le permiten mantener el acceso al sistema. Se pueden dividir en dos grupos los que van integrados en el núcleo o kernel y los que funcionan a nivel de aplicación. Los que actúan desde el kernel añaden o modifican una parte del código del kernel para ocultar el backdoor o puerta trasera. Normalmente este procedimiento se complementa añadiendo nuevo código al kernel, ya sea mediante un controlador (driver) en sistemas Windows o un módulo como los módulos del kernel de Linux. Estos rootkits suelen parchear las llamadas al sistema con versiones que esconden información sobre el intruso. Son los más peligrosos, ya que su detección puede ser muy complicada. Los rootkits que actúan como aplicaciones pueden reemplazar los archivos ejecutables originales con versiones crackeadas que contengan algún troyano, o también pueden modificar el comportamiento de las aplicaciones existentes usando hacks, parches, código inyectado, etc.

Esta lista podría ser el doble de larga, pero basta con conocer estos términos que son los más utilizados y que a veces no se manifiestan como variantes puras sino como una combinación de varias. Cualquier

profundización en otras terminologías puede realizarse en la bibliografía consultada o muchos sitios en Internet dedicados a esto y que incorporan nuevos términos que van surgiendo a diario.

1.2 Clasificación.

Los programas malignos tienen la única intención de hacer funcionar a los equipos de una manera que no es la adecuada o que no es la deseada por el usuario. Algunos programas malignos están diseñados para hacer daño, causando alteraciones en el disco duro, eliminando archivos o simplemente modificando ficheros, pero otros sólo se ocupan de replicarse o de hacer notar su existencia por medio de mensajes. Pero algo si está bien claro y es que frecuentemente provocan pérdida de información importante, fallos en el sistema operativo, o incluso, el colapso del mismo. Es por esto que se ha tratado a lo largo de los años de clasificar a los programas malignos para comprender como es que estos funcionan y de esta manera encontrar soluciones para evitarlos.

Los programas malignos se pueden clasificar de formas muy variadas. Una alternativa de agrupación sería dividirlos en dos grupos dependiendo de si se pueden reproducir, es decir, hacer copias de si mismos y extenderse o no. Dentro del primer grupo quedarían los virus y los gusanos, el resto no se puede reproducir.

Otro intento de clasificación sería de acuerdo al método de infección quedando divididos en: virus que infectan archivos, virus del sector de arranque, virus del sector de arranque maestro, virus múltiples y virus macro.

Virus que infectan archivos: Este tipo de virus normalmente infectan el código ejecutable, por ejemplo archivos .com y .exe. También pueden infectar otros archivos cuando se ejecuta un programa infectado desde un disquete, una unidad de disco duro o una red. Muchos de estos virus se mantienen residentes en memoria. Una vez que la memoria se infecta, cualquier archivo ejecutable que no esté infectado pasará a estarlo. Algunos ejemplos conocidos de virus de este tipo son Jerusalén y Cascade.

Virus del sector de arranque: estos virus infectan el área del sistema de un disco, es decir, el registro de arranque de los disquetes y los discos duros. Todos los disquetes y discos duros (incluidos los que sólo

contienen datos) tienen un pequeño programa en el registro de arranque que se ejecuta cuando se inicia el equipo. Los virus del sector de arranque se copian en esta parte del disco y se activan cuando el usuario intenta iniciar el sistema desde el disco infectado.

Estos virus están residentes en memoria por naturaleza. La mayoría se crearon para DOS, pero todos los equipos, independientemente del sistema operativo, son objetivos potenciales para este tipo de virus. Para que se produzca la infección basta con intentar iniciar el equipo con un disquete infectado. Posteriormente, mientras el virus permanezca en memoria, todos los disquetes que no estén protegidos contra escritura pueden quedar infectados al acceder a ellos. Algunos ejemplos de virus del sector de arranque son Form, Disk Killer, Michelangelo y Stoned.

Virus del sector de arranque maestro: estos virus están residentes en memoria e infectan los discos de la misma forma que los virus del sector de arranque. La diferencia entre ambos tipos de virus es el lugar en que se encuentra el código vírico.

Los virus del sector de arranque maestro normalmente guardan una copia legítima del sector de arranque maestro en otra ubicación. Los equipos con Windows NT infectados por virus del sector de arranque o del sector de arranque maestro no podrán arrancar. Esto se debe a la diferencia en la forma en que el sistema operativo accede a la información de arranque, en comparación con Windows 95/98. Si el sistema con Windows NT está formateado con particiones FAT, normalmente se puede eliminar el virus arrancando desde DOS y utilizando un programa antivirus.

Si la partición de arranque es NTFS, el sistema deberá recuperarse utilizando los tres discos de instalación de Windows NT. Algunos ejemplos de virus del sector de arranque maestro son NYB, AntiExe y Unashamed.

Virus múltiples: estos virus infectan tanto los registros de arranque como los archivos de programa. Son especialmente difíciles de eliminar. Si se limpia el área de arranque, pero no los archivos, el área de arranque volverá a infectarse. Ocurre lo mismo a la inversa. Si el virus no se elimina del área de arranque, los archivos que hayan sido limpiados volverán a infectarse. Algunos ejemplos de virus múltiples son One_Half, Emperor, Anthrax y Tequilla.

Virus macro: estos virus infectan los archivos de datos. Son los más comunes y han costado a empresas importantes gran cantidad de tiempo y dinero para eliminarlos. Con la llegada de Visual Basic en Microsoft Office 97, se puede crear un virus macro que no sólo infecte los archivos de datos, sino también otros archivos. Los virus macro infectan archivos de Microsoft Office: Word, Excel, PowerPoint y Access. Algunos ejemplos de virus macro son W97M.Melissa, WM.NiceDay y W97M.Groov.

1.3 Acciones más comunes que realizan los programas malignos.

A pesar de que el sistema antivirus tiene una cadena para identificar cada programa maligno y de esta manera diferenciar a unos de otros, la mayoría de los programas malignos presentan, en una que otra ocasión, comportamientos similares. Existen actividades que son realizadas frecuentemente por todos los programas, pero algunas específicas son raramente realizadas por el usuario común y pudieran ser seña de algún comportamiento maligno. Estas actividades generalmente vienen en correspondencia con un tipo de programa maligno determinado.

Entre las acciones más comunes realizadas por los programas malignos que se reproducen, resalta la búsqueda y copia recursiva a través de los ficheros ejecutables del disco duro. Aunque muchos programas malignos enmascaran su actividad de disímiles maneras como reproducirse sólo en un momento específico, o no reproducirse hacia ficheros determinados, estas acciones son detectadas fácilmente por la mayoría de los sistemas antivirus que basan su funcionamiento en detectar el acceso a los ficheros desde el modo kernel del sistema operativo detectando el programa maligno y de esta forma evitando la infección de nuevos ficheros.

De esta manera, a pesar de que la acción más común realizada por todos los virus y gusanos constituye el momento en que está intentando reproducirse o copiarse hacia una nueva locación, y dado que la mayoría de los sistemas antivirus ya tienen incorporados filtros en el sistema de ficheros, no constituye objetivo centrarse en esta acción debido a que ya posee mecanismo de detección. A continuación se analizarán los programas malignos partiendo de los objetivos que mayormente intentan llevar a cabo sus desarrolladores a fin de lograr identificar otras acciones que resulten de utilidad en su detección.

1.3.1 Garantizar la ejecución del código vírico.

Los programas malignos más sencillos tratan de buscar ficheros específicos a infectar, por ejemplo, los más usados, y de esta manera garantizar su ejecución cuando el usuario ejecute el programa hospedero dándole clic al mismo, o indirectamente al mandarse a ejecutar a través de otro programa que ya fue lanzado en ejecución anteriormente por el usuario, así esta cadena pudiera extenderse tanto como haya sido previsto por el programador del código maligno.

Por otra parte existen programas malignos un poco más complejos que en vista de lograr un mayor efecto en sus propósitos tratan de garantizar su ejecución el mayor tiempo posible, o el mayor número de veces. Para lograr esto la técnica más utilizada es la de lanzar la ejecución del código maligno cada vez que se inicie el sistema.

Existen un gran número de técnicas que permiten a los programas cargarse automáticamente cuando el sistema operativo Windows se inicia y que son usadas por programas de uso cotidiano. El método más sencillo es usar la carpeta del sistema *Startup* o *Inicio* (en dependencia del idioma del Sistema Operativo instalado). Si se utiliza este método el usuario tendrá la posibilidad de tener un poco de control sobre la ejecución, ya que de esta manera para evitar que se ejecute el programa maligno, el usuario solo tendría que borrar el fichero de esta carpeta. Este método también tiene, para el código maligno, la desventaja de que se ejecutará bajo los privilegios del usuario corriente, y si este no es un usuario del grupo de administración tendrá limitada su actividad. Los programas malignos generalmente intentan copiarse en la carpeta de inicio del usuario administrador, acción que completan fácilmente si el usuario que se encuentra iniciado en la máquina en el momento de la infección pertenecía al grupo de administración.

Un método más complejo es usar el registro de Windows, una base de datos utilizada por el sistema operativo para el manejo de los programas y del propio sistema. En el registro existen llaves o entradas utilizadas para indicar los programas que serán ejecutados junto con el sistema. Algunas de ellas son las siguientes:

HKCU\Software\Microsoft\Windows\CurrentVersion\Run: Un programa que se encuentre indicado en esta entrada será ejecutado cada vez que un usuario particular se inicie en el sistema.

HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce: Un programa que se encuentre indicado en esta entrada será ejecutado una vez que un usuario particular se inicie en el sistema y seguidamente será borrada.

HKLM\Software\Microsoft\Windows\CurrentVersion\Run: Un programa que se encuentre indicado en esta entrada será ejecutado cada vez que el sistema operativo este iniciando. No requiere que un usuario esté iniciado para ejecutarse, y corre bajo los privilegios de la cuenta del sistema.

HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce: Un programa que se encuentre indicado en esta entrada será ejecutado una vez que el sistema operativo este iniciando. No requiere que un usuario esté iniciado para ejecutarse, y corre bajo los privilegios de la cuenta del sistema. Seguidamente la llave es borrada.

HKLM\Software\Microsoft\Windows\CurrentVersion\RunServices: Esta entrada indica el lanzamiento de un servicio que correrá bajo la cuenta del sistema. Un ejemplo de servicio es el servicio instalado por los sistemas antivirus para interactuar con el sistema de filtrado de acceso a ficheros que constituye la base del motor antivirus.

HKLM\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce: Idéntica a la entrada anterior solo que es eliminada luego que se inicia la primera vez.

En caso de que el código maligno use el registro, eliminar su funcionamiento se hace un poco más complicado para los usuarios comunes, sobre todo para los que no estén familiarizados con la estructura del registro. Además, de esta forma el programa maligno puede contar con privilegios más altos que los que podría contar si utiliza la carpeta de inicio.

Como último, haciendo énfasis en las vulnerabilidades de seguridad que se cometen a diario, al programa maligno solo le es posible escribir en el registro si el usuario que se encuentra iniciado en la computadora en el momento de infección pertenece al grupo administradores, por esta razón es recomendable que si lo que se va a hacer en la máquina no requiere de los privilegios de administración, los usuarios no deberían

abrir una sección con privilegios tan altos ya que lo único que harían sería facilitarle el camino a los programas malignos.

1.3.2 Técnicas sofisticadas de ocultamiento y residencia en memoria.

Entre los objetivos perseguidos por los programas malignos no solo se encuentra garantizar su ejecución las veces que así lo requiera sino ocultar su funcionamiento e incluso prolongar su actividad el mayor tiempo posible. Resulta evidente para un usuario detectar procesos que no han sido ejecutados por el mismo, basta con ejecutar el administrador de programas y echar una mirada a los procesos. Con un poco de experiencia se puede estar familiarizado con los procesos del sistema y los que comúnmente se mantienen activos. No obstante aunque un programa maligno no se oculte de la lista de procesos, puede resultar enormemente dañino sobre todo si el método de propagación es bastante rápido.

El método más sencillo utilizado para mantener la ejecución el mayor tiempo posible es cuando en el código vírico se le introduce un ciclo infinito o lo suficientemente grande para que se logren los objetivos del programa maligno. Esto es detectado de manera fácil ya que el proceso del programa maligno consume un considerable tiempo de procesador que es revertido en lentitud no usual en los trabajos que se realizan a diario. Una de las estrategias utilizadas para evitar que surjan sospechas es utilizar un evento timer ó temporizador, mediante el cual el código vírico tomará el control del procesador a intervalos de tiempo regulares. Cada vez que se ejecuta el código vírico este chequeará algún evento por el que está esperando para lograr sus objetivos o sencillamente intentará reproducirse.

En los sistemas operativos es típico encontrar servicios brindados por el sistema que permiten detectar acciones realizadas por los programas que corren en él. En Windows existen diferentes métodos de detectar acciones del sistema operativo tanto en modo usuario como en modo kernel. Esta funcionalidad es conocida por los programadores como ganchos o hooks, su uso debe ser extremadamente cuidadoso y son muy usados también por los programas malignos.

Mediante un hook, el programa maligno puede tomar la ejecución solo cuando el evento deseado ocurra, y de esta manera, pasar desapercibido la mayor parte del tiempo. Por otra parte, un hook es un mecanismo

muy común para interceptar una sección particular de un código en ejecución para así modificar su comportamiento. Un hook que intercepta la llamada del sistema que retorna la lista de procesos ejecutándose en un instante dado constituye un ejemplo claro del proceso descrito.

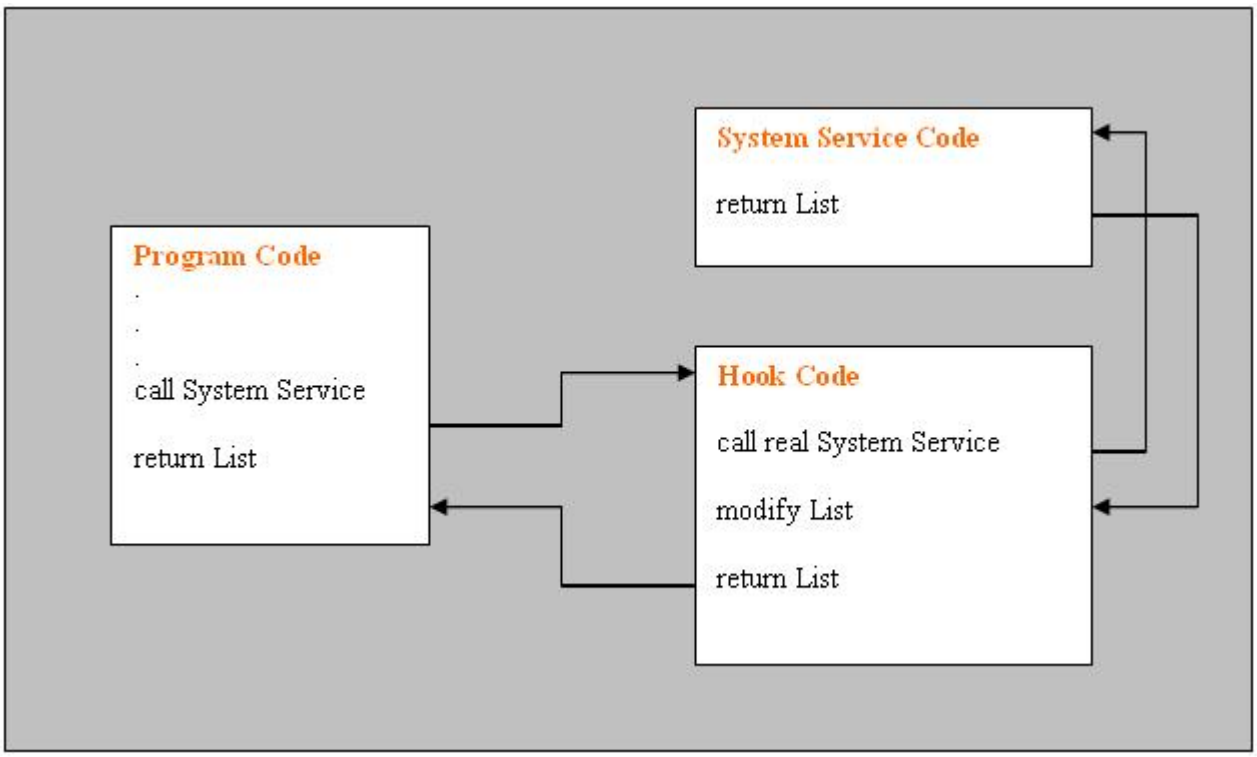

Figura 1.1. Ejemplo de un sistema con un hook instalado.

Cuando se realice la llamada al servicio del sistema encargado de retornar la lista de procesos en vez de esta se ejecutará el código del hook, el cual entonces realizará la llamada al verdadero servicio del sistema. El valor de retorno del servicio del sistema contendrá la lista de procesos, de la cual el código del hook eliminará el nombre del proceso maligno y posteriormente retornará la lista cambiada al proceso que realizó la llamada originalmente (figura 1.1). Este es un método utilizado por los programas malignos para ocultar su funcionamiento y que resulta muy difícil de ser detectado, es una estrategia compleja y como se mencionó anteriormente es utilizada por los rootkits integrados en el kernel.

Otra estrategia utilizada para ocultar el código vírico, es utilizar el espacio de memoria virtual de otro proceso para que el código maligno sea ejecutado desde otro proceso en la memoria y no desde el original que le dio vida. En el sistema operativo Windows cada proceso se ejecuta en su propio espacio de memoria virtual pero utilizando las funciones adecuadas del sistema se puede lograr lo que se conoce como inyección de código, la cual consiste en escribir en el espacio de memoria virtual de un proceso blanco a fin de modificar su código, añadir librerías de enlace dinámico o crear incluso nuevos hilos de ejecución. Esta técnica es una de las más peligrosas conocidas, y combinadas con algún método de reproducción rápido puede resultar en graves daños; tal fue el caso del gusano CodeRed que explotaba una vulnerabilidad de desbordamiento de buffer en el servicio de información de Internet de Windows que le facilitaba reproducirse por la memoria de las computadoras vulnerables, y como consecuencia, prácticamente no existía como fichero en el disco duro, lo que lo hizo muy difícil de detectar y descontaminar.

1.3.3 Técnicas de ataque.

Los programas malignos también utilizan técnicas de ataque como terminar los procesos que les impiden realizar sus acciones malignas o incluso, en ocasiones, terminar programas es simplemente el efecto maligno en si. Entre los procesos que son atacados con mayor frecuencia se encuentran los antivirus y los cortafuegos.

Constituye una actividad altamente sospechosa que un proceso termine su ejecución de manera anormal y más si fue terminado por otro proceso por lo que aparte de la protección que pueda tener un sistema antivirus como defensa ante ataques malignos de esta índole, es conveniente de alguna forma detectar este tipo de actividad como método de prevención.

Conclusiones.

Durante este capítulo se ha realizado un bosquejo del funcionamiento general de los programas malignos, enfatizando en aquellas acciones sospechosas que resultan factibles de interceptar como medida preventiva. A continuación se enuncian de forma separada con el objetivo de facilitar su posterior estudio.

- Intento de escritura en zonas peligrosas del registro de Windows.
- Intento de escritura en el espacio de memoria virtual de otro proceso.
- Intento de terminación anormal de procesos.

En los capítulos posteriores se parte de estos resultados y se proponen alternativas para la detección e intercepción de estas acciones. En algunos casos se contará con métodos otorgados por el mismo sistema para interceptarlas antes de que esa tarea sea completada, en otros sin embargo será necesario entrar en el mundo del hacking para lograr este objetivo, y serán necesarias muchas pruebas antes de contar con una solución verdaderamente confiable.

CAPÍTULO 2. INTERCEPCIÓN DE POSIBLES ACCIONES MALIGNAS.

Introducción.

En este capítulo se analizan las técnicas de intercepción para las posibles acciones malignas que se señalaron en el capítulo anterior. Se realiza un estudio del proceso de llamada a los servicios del sistema y específicamente los utilizados en la mayoría de los casos por los programas malignos. Muchos de los métodos utilizados requieren un alto conocimiento del comportamiento del sistema y de la programación a bajo nivel en el kernel del mismo, que van más allá del espectro de este trabajo pero que se pueden encontrar en la bibliografía consultada. Se concluye enunciando los métodos propuestos para la detección de acciones malignas.

2.1 Proceso de llamada a los servicios del sistema.

En todo sistema operativo existen un conjunto de funciones básicas las cuales constituyen el nivel inferior de todo el funcionamiento del sistema. Estas funciones son llamadas como servicios del sistema y no importa cuan avanzado sea el entorno de trabajo en el que se esté desarrollando un software siempre el flujo de ejecución termina en la llamada al servicio correspondiente. Los servicios del sistema forman parte del kernel del sistema operativo y abstraen al usuario de complicadas acciones sobre el hardware y el sistema operativo en general.

En el sistema operativo Windows a partir de la tecnología NT, todo proceso en el sistema puede ejecutarse en dos modos, conocidos como modo usuario y modo kernel o también nombrados a veces como modo poco privilegiado y modo privilegiado, respectivamente. Estos privilegios de ejecución no tienen nada que ver con los privilegios que encontramos en modo usuario, que son posibles de configurar y que mayormente dependen del nivel de privilegio del usuario corriente, sino que constituyen las capas del sistema operativo en si. Todo con lo que se interactúa directamente y a diario, pertenece al modo usuario, mientras que el modo kernel permanece desapercibido incluso para programadores que llevan una vida de experiencia realizando programas para este sistema.

Si se desea obtener el nivel de privilegio del kernel entonces es necesario diseñar un system device driver (dispositivo manejador del sistema). Los drivers como comúnmente se les llama son pensados en su mayoría como controladores de hardware pero en realidad estos presentan una arista fundamental que permite realizar operaciones imposibles desde modo usuario.

2.1.1 Librerías de enlace dinámico.

Cada vez que se ejecuta una acción sobre el sistema se está ejecutando intrínsicamente un conjunto de servicios en el modo kernel. Los programas en modo usuario utilizan un conjunto de funciones que usualmente llevan un nombre similar al del servicio que solicitan pero que abstraen al programador del código ensamblador que realiza la llamada al servicio (shellcode). Cada función o API (Application Program Interface) como también se le suele llamar se encuentra a su vez en una librería de enlace dinámico (dll), constituyendo éste, un método muy utilizado para la reutilización de código común. Tal vez la librería más conocida para la mayoría de los usuarios y programadores, ya que es la que radica en el modo poco privilegiado, es la librería ntdll.dll, a la cual van dirigidas todas las llamadas o invocaciones de funciones desde las otras DLL de propósitos específicos. Entre estas otras librerías las más comunes son:

- **User32.dll:** alberga las funciones de 32 bits de interacción con el usuario.
- **Gdi32.dll:** contiene todas las funciones para manejar la interfase grafica del sistema.
- **Kernel32.dll:** funciones para la operación con archivos y gestión de memoria.
- **Advapi32.dll:** utilizada para la autentificación y accesos al registro.
- **Winmm.dll:** encargada de la parte multimedia.
- **Comdlg32.dll:** maneja los controles más comunes de las aplicaciones de ventana.

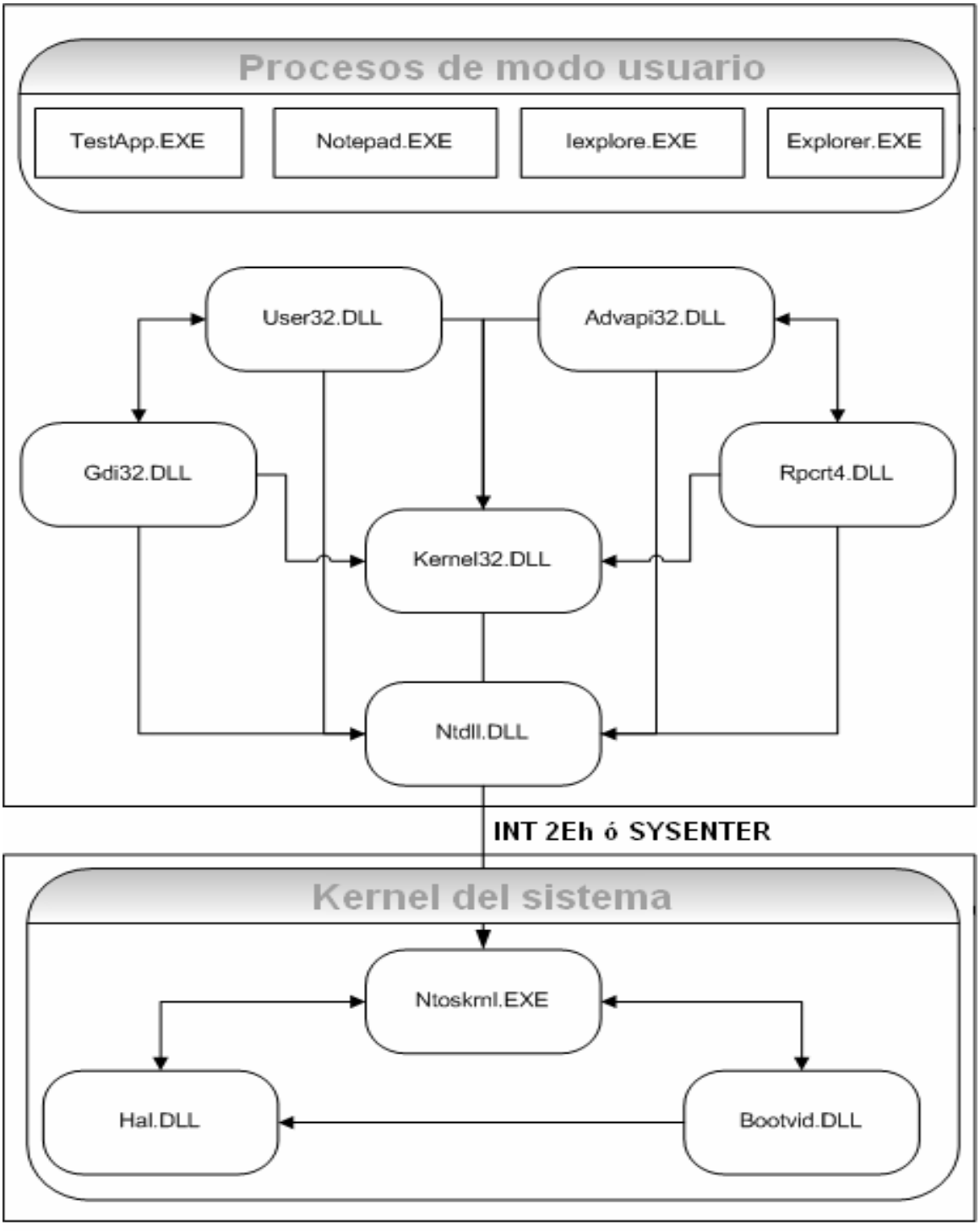

Figura 2.1. Esquema de llamadas a servicios del sistema operativo.

La mayoría de las llamadas a funciones que radiquen en estas u otras librerías serán dirigidas a la correspondiente función en la librería del sistema ntdll.dll, la cual es la encargada de llamar a los servicios correspondientes en modo kernel que completarán la petición o función invocada por el proceso de modo usuario (Figura 2.1).

En realidad existe un grupo de servicios que a pesar de no estar reflejados en la mayor parte de la bibliografía, no son atendidos por ntoskrnl.exe sino que los atiende el driver del sistema Win32k.sys. Entre estos servicios se encuentran la mayoría de las funciones que realizan el trabajo con gráficos.

El salto al modo kernel se realiza a través de la llamada a una interrupción del microprocesador (interrupt gate). A esta interrupción en el sistema operativo Windows con tecnología NT previo a las versiones de XP le corresponde el número 2Eh.

2.1.2 Entrada al kernel del sistema.

Si utilizando un depurador se establece un punto de ruptura en una API nativa del sistema operativo, podemos observar como ocurre el salto al modo kernel:

```
B814000000 mov   eax, 20h; NtCreateFile ID en Windows 2000
8D542404   lea   edx, [esp+arg_0]
CD2E       int   2Eh
C22C00     ret   2Ch
```

Previamente al llamado a la interrupción se pone el número del servicio solicitado en el registro eax y se apunta con edx a los parámetros pasados por medio de la pila a este servicio. El microprocesador busca en la tabla de descriptores de interrupción (IDT), en la entrada correspondiente al número 2Eh, el descriptor de compuerta de interrupción (IGD) que describe como la interrupción debe ser manipulada. Entre otras cosas aquí reside el segmento en que se encuentra la rutina de servicio de interrupción y el desplazamiento de esta en el segmento. Esta misma rutina de servicio es la encargada de verificar cada vez que se realiza una llamada al kernel el valor del registro eax, con el fin de ejecutar el servicio particular

solicitado. También es la encargada de copiar los parámetros de la pila de modo usuario a la pila de modo kernel.

En Windows XP y las versiones posteriores se hace uso de una nueva instrucción de los procesadores de Intel llamada SYSENTER, que fue incorporada a partir del Pentium II, la cual permite ahorrarse varios ciclos de CPU en el cambio de nivel de operación. Esta instrucción utiliza el echo de que cada vez que se realiza una llamada del sistema siempre se dirige el flujo de ejecución a la misma rutina de despacho, entonces se codifica por hardware la dirección de salto. El resto del trabajo es bastante similar ya que se usan las mismas antiguas estructuras que se utilizaban con la interrupción 2Eh.

El llamado al modo kernel ahora resulta como se muestra:

```
B827000000  mov    eax, 27h; NtCreateFile ID en Windows XP
BA0003FE7F  mov    edx, 7FFE0300h
FFD2        call  [edx]
C20C00      retn   0Ch

CALL EDX -> (7FFE0300 – Cerca del fin del espacio de direcciones del usuario)

8BD4        mov edx, esp
0F34        sysenter
C3          retn
```

Paralelamente al desarrollo de SYSENTER en la arquitectura Intel de 32 bits, o lo que es lo mismo IA32, AMD desarrolló la instrucción SYSCALL, la cual tiene un desempeño similar a SYSENTER para los sistemas operativos Windows XP y posteriores.

El sistema operativo no presenta ficheros binarios diferentes para los tres casos de llamadas al sistema vistos anteriormente sino que añade una página en el entorno de cada proceso en la que se encuentra una función llamada *SystemCallStub()* encargada de realizar el llamado al método que corresponda.

2.2 Intercepción en las llamadas a servicios del sistema.

Una vez analizado el proceso de llamada a los servicios del sistema entonces queda por detectar el momento en todo este trayecto antes de que sea llevada a cabo la acción en el que es más seguro y

eficiente realizar la intercepción. Este mecanismo es conocido como hooking y es muy aplicado en los sistemas operativos con fines de seguridad y registro.

Existen dos diferentes mecanismos de hooking en cuanto al nivel o modo de operación en que son implementados.

2.2.1 Hooking en modo Kernel.

Durante la inicialización del kernel del sistema este crea una tabla de funciones, que después será referida como la tabla de despacho de servicios del sistema (SSDT), para los diferentes servicios que provee. Cada entrada en la tabla contiene la dirección de una función dado el identificador del servicio (Figura 2.2).

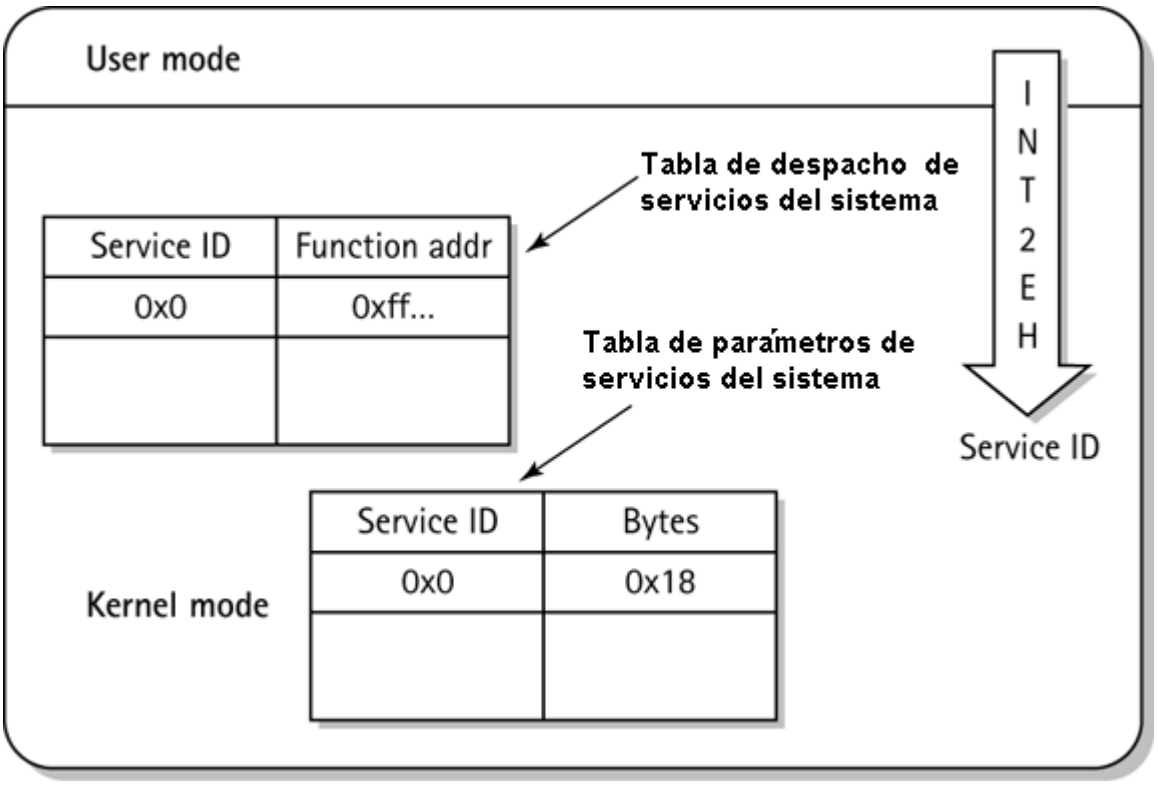

Figura 2.2. Tablas de despacho y de parámetros de servicios del sistema.

El manipulador de la interrupción del sistema busca en esta tabla la dirección de la función correspondiente. El código para cada servicio también reside en el kernel. Similarmente otra tabla llamada la tabla de parámetros de servicios del sistema (SSPT) provee al manipulador de la interrupción con el número de parámetros necesarios para cada función.

La forma más sencilla de hacer hooking en modo kernel es localizar la SSDT usada por el sistema operativo y cambiar el puntero a la función por otra función insertada por el desarrollador. Esto solo es posible realizarlo desde un driver de modo kernel ya que esta tabla es protegida por el sistema operativo a nivel de tabla de páginas.

Además de la tabla de servicios del sistema existe a su vez otra tabla conocida como tabla de despacho de servicios del sistema a la sombra (SSDTS), que es creada en la inicialización del driver del sistema Win32k.sys. Paralelamente también existe su correspondiente tabla de parámetros de servicios del sistema a la sombra (SSPTS). Estas tablas son utilizadas para el manejo de los servicios encargados de llevar a cabo el funcionamiento gráfico del sistema.

Debido a la necesidad constante de los desarrolladores de productos de seguridad, Microsoft ha documentado funciones que permiten enlazar acciones del sistema operativo desde modo kernel. Entre este grupo de acciones se encuentran los accesos al registro, y la carga y descarga de módulos, aunque en estas ultimas solo se soporta la notificación y no la intercepción de la acción.

2.2.2 Hooking en modo usuario.

Este tipo de intercepciones es logrado a través de una librería de enlace dinámico que es inyectada en el contexto de cada proceso. También es conocido el método de sobre escritura de las instrucciones encargadas de realizar el llamado a las funciones que se desean interceptar en el proceso objeto. La intercepción es fácil de ser evitada ya que basta que el proceso maligno realice el llamado directamente al modo kernel y no a través de las APIs nativas del sistema para que todo mecanismo de seguridad basado en este tipo de hook sea burlado.

Desde el modo usuario Microsoft también ha documentado un conjunto de acciones que permiten ser enlazadas a través de la API *SetWindowsHookEx()*. No obstante este grupo de acciones están más dirigidas hacia el enfoque del entorno grafico del sistema, y no abarcan acciones sobre los procesos o el registro.

2.3 Proceso de escritura en el registro.

En el capítulo anterior se evidenció que una de las acciones que más denotan la presencia de un programa maligno es la escritura en el registro sin conocimiento del usuario corriente.

La función o API del sistema que permite escribir un dato de un tipo determinado en una llave del registro dada es *RegSetValueEx()*. Esta función presenta el prototipo siguiente:

```
LONG RegSetValueEx (
                     HKEY hKey,
                     LPCTSTR lpValueName,
                     DWORD Reserved,
                     DWORD dwType,
                     const BYTE* lpData,
                     DWORD cbData
                    );
```

Donde el parámetro *lpData* indica el valor que se desea escribir en el registro, pero sin duda el más importante es *hKey* el cual es el manipulador (handle) a la llave específica en que se desea escribir. Este handle es obtenido a su vez con la API *RegOpenKeyEx()*. El prototipo se muestra a continuación:

```
LONG RegOpenKeyEx(
                   HKEY hKey,
                   LPCTSTR lpSubKey,
                   DWORD ulOptions,
                   REGSAM samDesired,
                   PHKEY phkResult
                  );
```

En esta el puntero al handle que será utilizado en la llamada a *RegSetValueEx()* es devuelto en el parámetro *phkResult.* En la librería de desarrollo de Microsoft, MSDN, se puede encontrar mucha más información sobre el uso de estas funciones.

Seguidamente al llamado a *RegSetValueEx()* el flujo de ejecución en el sistema sigue el curso hasta el servicio del sistema que corresponda en el kernel del sistema operativo.

2.4 Intercepción de accesos al registro de Windows desde el modo Kernel.

El método documentado y con más altos niveles de seguridad para realizar la intercepción de posibles acciones malignas sobre el registro, es realizar un hook de modo kernel a través de la función *CmRegisterCallback()* dada por Microsoft para este propósito.

En lo posterior se abordarán algunos aspectos a tener en cuenta en el diseño de un driver capaz de interceptar los accesos al registro de Windows pero sin ahondar en particularidades sobre el diseño general de un driver.

La función *CmRegisterCallback()* está disponible para los sistemas operativos Windows XP y los posteriores, aunque la documentación recomienda que en Windows Vista se use la versión extendida de esta función. En lo que respecta a este trabajo se enfatizará en la versión original, de la cual será probado su funcionamiento con el mecanismo de intercepción propuesto.

Un driver que llame a *CmRegisterCallback()* lo que hará será registrar una rutina de *RegistryCallback* la cual será llamada cada vez que se realice una operación en el registro para monitorearla, bloquearla o cambiarla.

Un punto a tener en consideración a la hora de implementar cualquier trabajo en modo kernel es el privilegio de interrupción bajo el cual se ejecutará el código. El procesador esencialmente ayuda a determinar quien puede ser interrumpido en un momento determinado o no, para ello existen los IRQL (Interrup ReQuest Level o Nivel de Demanda de Interrupción). Esto posibilita que un hilo pueda ser interrumpido por un código que necesita correr solo si este código tiene un mayor nivel de IRQL en el mismo procesador. Los niveles de IRQL son PASSIVE_LEVEL el cual es el más bajo, APC_LEVEL, DISPATCH_LEVEL y DIRQL (Device IRQL) que es el más alto.

Por lo menos la sección de código donde se hace la llamada a *CmRegisterCallback()* debe correr a un nivel de IRQL de APC_LEVEL. El prototipo de la función es el siguiente:

```
NTSTATUS  CmRegisterCallback(
                            IN PEX_CALLBACK_FUNCTION Function,
                            IN PVOID Context,
                            OUT PLARGE_INTEGER Cookie
                            );
```

Donde *Function* no es más que el puntero para registrar la rutina *RegistryCallback()*, *Context* es un puntero a una zona de memoria donde se encuentran los datos que se desea que estén disponibles en el momento que el sistema operativo realice la invocación del callback y *Cookie* es un identificador que debe ser almacenado para la desinstalación del callback.

Un callback no es más que un mecanismo brindado por el sistema para que el programador del software pueda registrar una función creada por el mismo en una especie de cadena o lista para que sea llamada en respuesta a un evento determinado.

La rutina *RegistryCallback()* provee a los componentes del modo kernel una vía de obtener el control de cualquier aplicación que esté accediendo al registro. Debe ser registrada en la cadena de callbacks de intercepción del registro antes de que cualquier acceso sea realizado, pues solo de esta manera es que podrá interceptar la operación que tome lugar. Esta rutina está disponible en Windows XP en adelante. Debe correr al nivel de IRQL PASSIVE_LEVEL y siempre será invocada en el contexto del hilo que está realizando la operación en el registro, lo que significa que el proceso activo en el espacio lógico del usuario será el mismo que está intentando acceder al registro. Esta rutina está basada en el mecanismo general de callbacks definido en el fichero cabecera para el diseño de drivers ntddk.h. A continuación se muestra el prototipo de la función *RegistryCallback()*.

```
NTSTATUS   RegistryCallback(
                           IN PVOID CallbackContext,
                           IN PVOID Argument1,
                           IN PVOID Argument2
                           );
```

A pesar de la gran cantidad de APIs del registro que existen, el sistema operativo usa este simple callback para todas las APIs y esto es mucho mejor que tener un callback distinto para cada API. Es una opción para el programador realizar un tratamiento diferenciado en el interior de la rutina callback para cada una de las operaciones realizadas sobre el registro.

Una diferencia fundamental entre los parámetros pasados a las APIs del registro y los parámetros pasados al correspondiente procedimiento callback es que al procedimiento callback se le pasa un puntero a un objeto en lugar del handle a través del cual el usuario está realizando la operación. Esto ocurre de esta manera pues no es seguro usar handles en los calllback ya que pueden ser cerrados y rehusados de manera asincrónica con los callbacks. Los punteros a objetos evitan este problema ya que toman la referencia en el objeto antes de pasar el puntero al callback, y se libera la referencia después que el callback retorna, pues el callback no debe mantenerse con la referencia del mismo. El uso de punteros a objetos del kernel esta reservado solo para los drivers mientras que en modo usuario solo es posible interactuar con estos objetos a través de un handle y el objeto se presenta entonces como una zona de memoria opaca a la que solo se puede acceder por medio de funciones proporcionadas por el sistema.

Para cada operación del registro, la rutina callback será invocada dos veces una vez antes de que la operación sea realizada y otra luego de que sea realizada. .El sistema provee dos tipos de notificaciones: pre-notification y post-notification.

Pre-notificación: una llamada de este tipo a *RegistryCallback* indica que la operación del registro será encuestada, o sea, el sistema pasará los parámetros de la operación en el parámetro *Argument2*. La rutina callback puede decidir el resultado de una operación en el registro y el sistema usará el valor de retorno de la rutina callback como el valor de retorno de la operación para el usuario. Si un callback retorna un error el sistema ni siquiera intentará el acceso al registro. El seudo código para la operación realizada por el sistema en una pre-notificación es el que sigue:

```
if ( AreCallbacksRegistered() )
{
  for (todos los callbacks registrados)
  {
```

```
    status = callback(…);
    if ( !NT_SUCCESS(status) )
    {
       //retorna al estado de quien realizó la llamada a la API
       //sin hacer nada en el registro
       return status
    }
  }
}
```

Post-notificación: una llamada de este tipo a *RegistryCallback* indica que el sistema está finalizando una operación en el registro. El sistema ignora el valor de retorno de la rutina callback. El programador puede comprobar el valor del identificador del hilo que realiza la operación almacenando este en la pre-notificación y chequeando después en la post-notificación.

La rutina callback internamente puede acceder al registro pero ningún callback será ejecutado producto de estos accesos.

Luego de conocer todos los detalles de la rutina *RegistryCallback* se puede pasar a ver un segmento de código de cómo luciría una simple rutina callback:

```
NTSTATUS RegistryCallback(
                        IN PVOID CallbackContext,
                        IN PVOID Argument1,
                        IN PVOID Argument2
                        )
{
      REG_NOTIFY_CLASS Type;
      Type = (REG_NOTIFY_CLASS)Argument1;
      switch( Type ) {
      case RegNtPreCreateKeyEx:
      {
            REG_ CREATE_KEY_INFORMATION pCreate =
            (REG_ CREATE_KEY_INFORMATION)Argument2;
            // Aquí va el código para manipular CreateKey
            //(creación de una llave)
      }
      break;
      case RegNtPreDeleteKey:
      {
            PREG_DELETE_KEY_INFORMATION pDelete
```

```
            = (PREG_DELETE_KEY_INFORMATION)Argument2;
            // Aquí va el código para manipular NtDeleteKey
            //(eliminar llave)
            //
      }
      break;
      case RegNtPreSetValueKey:
      {
            PREG_SET_VALUE_KEY_INFORMATION pSetValue
             = (PREG_SET_VALUE_KEY_INFORMATION)Argument2;
            //
            //Aquí va el código para manipular NtSetValueKey
            //
      }
      break;
      case RegNtPreDeleteValueKey:
      {
            PREG_DELETE_VALUE_KEY_INFORMATION pDeteteValue
            = (PREG_DELETE_VALUE_KEY_INFORMATION)Argument2;
            //
            // Aquí va el código para manipular NtDeleteValueKey
            //
      }
      break;
      default:
            //
            // Como no interesa hacer hook a otras APIs
            // se dejan pasar para que retorne STATUS_SUCCESS.
            //
      break;
      }
      return STATUS_SUCCESS;

}
```

Al igual que existe la función *CmRegisterCallback()* para registrar la rutina *RegistryCallback*, existe una función para desregistrarla, se llama *CmUnRegisterCallback()*. Esta función debe ser llamada desde el nivel IRQL APC_LEVEL y su prototipo es el siguiente:

```
NTSTATUS CmUnRegisterCallback(
                              IN LARGE_INTEGER Cookie
                              );
```

Donde *Cookie* identifica la rutina callback a desregistrar.

2.5 Estrategias para la inyección de código y la terminación anormal de procesos.

Otras de las acciones malignas referidas en el capítulo anterior fueron la inyección de código y la terminación anormal de procesos, que aquí serán analizadas en conjunto dado el alto grado de relación en cuanto al método de llevarlas a cabo.

Existen al menos tres métodos para la inyección de código que fueron analizados para este trabajo:

1. Poner el código a inyectar en una librería de enlace dinámico y luego cargar esta en el proceso objeto a través de *SetWindowsHookEx().* Esta función hace referencia al servicio *NTUserSetWindowsHookEx()* que se encuentra en el driver del sistema Win32k.sys y que es referenciado a través de la SSDTS.

2. Poner el código a inyectar en una librería de enlace dinámico y luego cargar esta en el proceso objeto a través de *CreateRemoteThread()* y *LoadLibrary().* La función que evidencia que este método se está llevando a cabo es *CreateRemoteThread()* la cual realiza el llamado al servicio *ZwCreateThread()* en ntoskrnl.exe referenciado a través de la SSDT.

3. Copiar el código a inyectar directamente en el proceso objeto a través de *WriteProcessMemory()* y *CreateRemoteThread()*. En este método se realiza el llamado a dos servicios del sistema, ambos en ntoskrnl.exe, *ZwWriteVirtualMemory()* y *ZwCreateThread().*

Para la terminación anormal de procesos, existen un gran número de alternativas.

1. Terminar el proceso usando *TeminateProcess().* El servicio llamado en ntoskrnl.exe es *ZwTerminateProcess().*

2. Terminar el proceso llamando a la función *TerminateThread()* por cada hilo presente en el proceso. El servicio llamado es *ZwTerminateThread().*

3. Terminar el proceso creando un nuevo hilo de ejecución en este que realice un llamado a *ExitProcess()*. El detalle aquí se encuentra en los métodos para la inyección de código vistos anteriormente.

4. Terminar el proceso escribiendo en su memoria virtual caracteres nulos, a fin de que el sistema lo cierre en el momento de fallo. Nuevamente se remonta a la inyección de código.

5. Terminar el proceso utilizando variantes de las anteriores, incluso variantes en modo kernel que llamen a los servicios del sistema directamente.

Existen un grupo de servicios del kernel que resultan de interés después de este estudio:

- *NtUserSetWindowsHookEx()* (win32k.sys)
- *ZwCreateThread()* (ntoskrnl.exe).
- *ZwWriteVirtualMemory()* (ntoskrnl.exe).
- *ZwTerminateProcess()* (ntoskrnl.exe).
- *ZwTerminateThread()* (ntoskrnl.exe).

Nuevos métodos de inyección de código son vistos a diario e incluso es aún mayor la razón con que aumentan los métodos para la terminación anormal de un proceso. No obstante es difícil encontrar una variante nueva que no utilice estos servicios vistos anteriormente.

2.6 Intercepción de inyección de código y terminación inadecuada de procesos.

A pesar de que el método de interceptar las llamadas al sistema es sumamente complicado y peligroso, ya que puede traer consigo el mal funcionamiento del sistema, muchos productos de seguridad de gran renombre internacional, como el Karspersky Antivirus, lo utilizan. El punto en realidad no radica en utilizar el método de intercepción de servicios del kernel o no, sino en utilizarlo adecuadamente, o sea, teniendo en cuenta todas las consideraciones que el mismo sistema tiene cuando maneja el servicio original.

El primer paso para establecer un hook en los servicios del sistema es localizar la tabla de despacho de servicios del sistema y cambiar el puntero a la función que se desee interceptar por la dirección de una función del desarrollador. Todo este trabajo solo se puede realizar desde modo kernel ya que las tablas de servicios así como los servicios del sistema pertenecen a este.

Existe una entrada indocumentada en la lista de exportaciones de ntoskrnl.exe llamada *KeServiceDescriptorTable()*. Esta entrada es utilizada por el sistema para acceder a la SSDT y la SSPT. La estructura de esta entrada se muestra a continuación:

```
typedef struct ServiceDescriptorTable{
                                    PVOID ServiceTableBase ;
                                    PVOID ServiceCounterTable(0);
                                    unsigned int NumberOfServices ;
                                    PVOID ParamTableBase ;
                                    };
```

Donde el primer parámetro como su nombre lo indica contiene la dirección base de la SSDT, el segundo parámetro indica el número de servicios en esta tabla, el tercero es solo usado en construcciones chequeadas del sistema operativo y contiene el número de veces que cada servicio es llamado; el último parámetro indica la dirección base de la SSPT.

ServiceTableBase y *ParamTableBase* contienen un número de entradas igual a *ServiceCounterTable*. Cada entrada representa un puntero a una función que implementa el correspondiente servicio.

Luego de encontrar la SSDT solo resta salvar la dirección de la función original del sistema, con el objetivo de que sea llamada posteriormente del chequeo, y sustituir la dirección original en la tabla por la dirección de la función hook.

Conclusiones.

Durante este capítulo se ha realizado un estudio de las acciones malignas identificadas en el capítulo anterior, detectando con este las llamadas a servicios del sistema que son realizadas con el fin de proponer mecanismos de intercepción en el modo kernel. En el caso del registro se cuenta con funciones

de intercepción documentadas para el sistema operativo, no siendo así para los otros servicios del sistema, para los cuales no queda otra opción que realizar la intercepción sustituyendo entradas en la tabla de servicios del sistema.

Los métodos resultantes del análisis son los siguientes:

1. Utilizar la función *CmRegisterCallback()* para registrar una función hook que intercepte las acciones de escritura en llaves peligrosas del registro. Para ello la función hook debe obtener el nombre de la llave a la que se está accediendo y verificar si es peligrosa o no. El driver entonces tomará la acción que se le haya predefinido o preguntará al usuario la acción a tomar. Todo esto es a consideración del diseñador del software de protección.

2. Realizar un hook en la SSDT para los servicios del sistema que se muestran a continuación:

   - *ZwCreateThread()* (ntoskrnl.exe).
   - *ZwWriteVirtualMemory()* (ntoskrnl.exe).
   - *ZwTerminateProcess()* (ntoskrnl.exe).
   - *ZwTerminateThread()* (ntoskrnl.exe).

   Para estos servicios se debe obtener en cada caso el proceso que realiza la acción y el proceso objeto sobre el que se ejecutará. El driver entonces debe tomar la acción que haya sido seleccionada por defecto o preguntar al usuario la acción a tomar. Todo esto es a consideración del diseñador del software de protección.

3. Realizar un hook en la SSDTS para el servicio del sistema que se muestran a continuación:

   - NtUserSetWindowsHookEx() (win32k.sys)

   Para este servicio solo será necesario detectar si el hook que desea instalar esta función en el sistema es de alcance global o local, siendo el primero el centro de atención en cuanto a seguridad.

El diseño de la comunicación entre el modo usuario y el modo kernel quedan a consideración del diseñador del software de seguridad.

CAPÍTULO 3. VULNERABILIDADES DETECTADAS.

Introducción.

Cada día las vulnerabilidades se hacen mayores y el riesgo de caer en ellas se incrementa casi sin que los usuarios, o los propios programadores, se den cuenta. Es por eso que en este capítulo se abordan los problemas que pueden hacer que los métodos mencionados en el capítulo anterior fallen o no sean del todo eficientes, debilitando de esta manera la defensa preventiva.

3.1 Tiempo de chequeo y tiempo de uso.

El problema Tiempo de chequeo y tiempo de uso o (Time of check to time of use) viene siendo conocido desde hace ya algún tiempo. Este problema ocurre cuando un proceso chequea una característica particular de un objeto y luego toma una acción asumiendo que la característica permanece todavía sin ser esto cierto, o sea que esta vulnerabilidad ocurre cuando un segundo evento depende de otro anterior.

Suponiendo que se cuenta con el siguiente algoritmo muy común en la práctica:

1. Obtener el nombre del objeto.
2. Chequear el nombre del objeto contra las políticas de seguridad.
3. Si el acceso es permitido entonces llamar al servicio del sistema original.
4. Si el acceso es denegado retornar un código de error.

Debido al hecho de que el nombre del objeto se encuentra en la memoria del usuario, el atacante puede cambiarlo entre el segundo y el tercer paso. De esta manera el atacante puede lograr que se llame a un servicio del sistema enviando primero un nombre de objeto para el cual no tiene el acceso restringido, que será usado por el proceso de seguridad para hacer la comprobación, pero inmediatamente el atacante cambia el nombre del objeto por uno restringido y obtiene el acceso.

Una variante que intenta no incurrir en la vulnerabilidad anterior es la que se muestra a continuación:

1. Llamar al servicio del sistema original para obtener el handle.
2. Obtener el nombre del objeto por medio del handle.
3. Chequear el nombre del objeto contra las políticas de seguridad.
4. Si el acceso está permitido retornar que todo está correcto.
5. Si el acceso es denegado cerrar el handle al objeto y retornar un código de error.

Este método no pasa de ser un intento que complica un poco las cosas para el atacante pero que no proporciona una solución definitiva. En este caso el atacante puede usar el handle para acceder al objeto entre los pasos dos y cuatro, al utilizar la peculiaridad del sistema de siempre otorgar los mismos valores de handle cada vez que se inicie el programa atacante.

Un ejemplo sencillo de la vulnerabilidad tiempo de chequeo a tiempo de uso es un programa que protege un fichero determinado donde este programa no debe alterar el fichero a menos que el usuario tenga los privilegios para hacerlo. El segmento de código se muestra a continuación:

```
if (access(filename, W_OK) == 0)
{
 if ((fd = open(filename, O_WRONLY)) == NULL)
 {
   perror(filename);
   return(0);
 }
 /*aquí va el código para modificar el fichero*/
}
```

Si el objeto referenciado por *filename* cambia entre las dos llamadas al sistema (la llamada *access(filename, W_OK)* y la llamada *open(filename, O_WRONLY)*) el segundo objeto será llamado en

open() sin comprobarse el acceso pues ya se había comprobado, aunque fue para el primer objeto. La figura 3.1 muestra el comportamiento descrito.

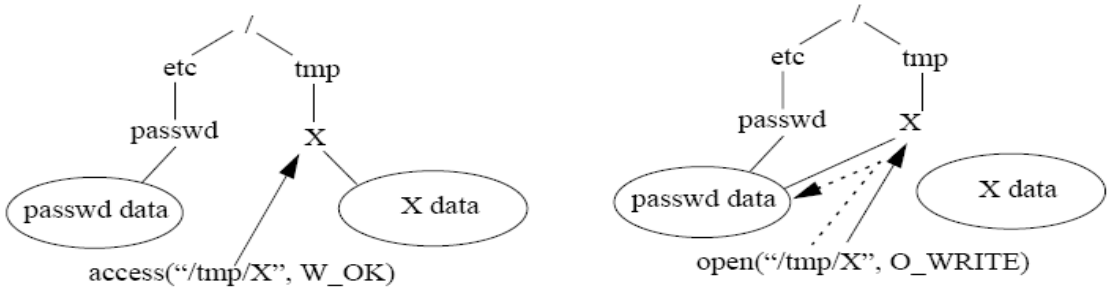

Figura 3.1. Ejemplo de la vulnerabilidad: Tiempo de chequeo a tiempo de uso.

La figura muestra claramente como en el momento en que se hace la llamada al sistema *access("/temp/X", W_OK)*, /temp/X y /etc/passwd estaban referenciando a dos objetos diferentes. Sin embargo justo en el momento antes de realizarse la llamada al sistema *open("/temp/X", O_WRITE)* se deja de referenciar al objeto y comienza a referenciar al mismo que /etc/passwd, como muestra la flecha con líneas discontinuas, de esta manera el objeto que es abierto no tiene nada que ver con el que se quería abrir, trayendo consigo que el fichero que es abierto no se le comprueba el acceso.

La raíz de este problema radica en que el chequeo es realizado contra recursos accesibles desde el modo usuario. Para mitigar estos errores el proceso que realiza el chequeo de seguridad debe copiar del modo usuario todos los buffers y sub-buffers a la memoria del modo kernel, tarea que a veces puede resultar en extremo complicada.

3.2 Restauración de la SSDT.

Una de las maneras de burlar la protección establecida mediante los hooks es intentar restaurar la tabla de despacho de servicios a sus valores originales y de esta manera las funciones del proceso de seguridad no serán llamadas nuevamente. La SSDT existe en el modo kernel, y normalmente para modificar una entrada en la misma, un programa debe cargarse a si mismo como un driver en modo

kernel. Sin embargo es posible desde un programa en modo usuario modificar esta tabla escribiendo directamente en la memoria del kernel a través del objeto del sistema *device**physicalmemory*.

La siguiente secuencia de pasos describe como un programa de modo usuario que corre con privilegios de administración puede obtener accesos de escritura y lectura (read/write access) en la memoria del kernel usando *device**physicalmemory*.

1. Abrir un objeto sección con los privilegios para escribir y leer activados, o sea, SECTION_MAP_READ | SECTION_MAP_WRITE para de esta manera obtener un handle a *device**physicalmemory*. Esto usualmente fallará, debido a que el administrador tiene solo derecho de lectura a *device**physicalmemory*.

2. Abrir un objeto sección con privilegio de READ_CONTROL | WRITE_DAC obteniendo un handle a *device**physicalmemory* permitiendo así escribir una nueva lista de control de acceso que será adicionada al objeto *device**physicalmemory*.

3. Adicionar la lista de control de acceso a \device\physicalmemory garantizando el privilegio SECTION_MAP_WRITE a la cuenta de administración.

4. Tratar de obtener un handle a *device**physicalmemory* nuevamente con los privilegios SECTION_MAP_READ | SECTION_MAP_WRITE los cuales ya deben estar permitidos.

Luego de realizar esta secuencia de pasos un programa que corra en modo usuario podrá obtener un handle a *device**physicalmemory*. Para escribir en la memoria física, el programa tendrá que mapear la página de memoria física de la SSDT en su espacio de memoria virtual.

Una vez mapeada la página de memoria física en el espacio de memoria virtual del proceso se hace una comparación con la tabla de despacho de servicios original y si se encuentra alguna discrepancia significa que se le está haciendo hook a algún servicio del sistema y el atacante podrá reestablecer la tabla con la original.

La restauración de la tabla de despacho de servicios del sistema constituye una vulnerabilidad, si se parte del hecho de que puede ser usada por atacantes para restaurar la tabla y eliminar los hook que se hayan instalado para proteger ciertos accesos, pero a la vez si es usada por programadores con buenas intenciones puede utilizarse para detectar cuando se está ejecutando un rootkit.

3.3 Validación de parámetros.

Un parámetro que es proporcionado desde el modo usuario y que se usa en el modo kernel sin ser validado, constituye un hueco de seguridad que también puede ser aprovechado. Cada vez que ocurre una llamada a una función desde el modo usuario, esta se traduce a una llamada en el modo kernel a la función correspondiente. Cada vez que esto ocurre, se hace una comprobación de parámetros pues en el modo kernel, por seguridad, no se confía en los parámetros enviados desde el modo usuario ya que pueden estar comprometidos por programas malignos.

Si un desarrollador realiza un hook de modo kernel a una función determinada lo que hace literalmente es ponerse en medio de las dos funciones, la de modo usuario y la de modo kernel, con la particularidad de que el hook se encuentra también en modo kernel. Cuando se termine de hacer todo el análisis que tenga implicado el hook y se haga la llamada a la función en modo kernel, el modo previo (modo anterior o modo del que se realizó la llamada) será el modo kernel, y por lo tanto no se realizará comprobación alguna de los parámetros ya que el modo kernel confía en todo lo que viene desde el propio modo kernel.

Es de extremada importancia realizar la validación de parámetros con la misma rigurosidad que la realiza el sistema, de lo contrario se puede incurrir en pérdidas de la estabilidad o incluso de la caída total de este.

3.3.1 Validación impropia de punteros del modo usuario.

Como es conocido, existen 2GB de espacio de direcciones de memoria para el modo kernel y 2GB para el modo usuario, pero esto no siempre es así, por ejemplo, el sistema puede ser configurado para que tenga 3GB en el modo usuario y 1GB en el modo kernel, lo que puede hacerse fácilmente a través del fichero

boot.ini. Si un desarrollador intenta determinar si una dirección del modo usuario es válida o no comparando directamente con el mayor número que generalmente puede tener el modo usuario cuando terminan los 2GB, puede ser que este incurra en un error debido a esta suposición.

Para evitar esto, en la validación de punteros de modo usuario debe usarse alguna función proporcionada por el sistema que devuelva si la dirección es válida o no. Esta función por el simple hecho de ser proporcionada por el sistema tendrá en cuenta los cambios que hayan sido realizados en el mismo. En resumen, no se puede dar por sentado que algo está correcto porque sea lo que trae por defecto el sistema ya que el propio sistema brinda facilidades para realizar cambios.

3.3.2 Validación impropia de objetos del modo kernel.

Windows otorga una serie de facilidades a través de los objetos del modo kernel, pero en estos objetos solo se puede influir desde el modo usuario a través de los handlers. En el modo kernel todos los handlers comparten el mismo espacio de nombres para los diferentes objetos. A causa de esto, un trabajo importante a realizar, es comprobar si el objeto al que se está referenciando un handle dado, es el esperado.

Es importante cada vez que se tiene un handle en el modo kernel y previendo que un handle puede perder el valor, e incurrir en un error grave en el kernel, tratar de obtener un puntero al objeto haciendo una traducción del handle.

3.4 Desconocimiento del usuario.

No se hace nada con establecer una defensa preventiva que proteja de las acciones más comunes realizadas por los programas malignos si cada vez que se le notifique al usuario de un evento sospechoso este no sepa que hacer o tome una decisión al azar pudiendo ser esta una decisión errónea.

Es común encontrar usuarios que para hacerse la vida más fácil solo se dedican a dar su consentimiento a todo los que el sistema pone ante ellos. La mayor vulnerabilidad que pude tener un sistema de defensa preventiva es la poca cultura de seguridad del usuario.

Conclusiones.

Durante este capítulo se han mencionado los errores en que usualmente se incumbe cuando se realiza el tipo de software que se está proponiendo como método de defensa preventiva y que no pueden ser tomados a menos ya que existen potenciales atacantes con todo el tiempo del mundo y con las herramientas necesarias para realizar un estudio de cada una de las particularidades del sistema desarrollado, con el afán de poder explotar las vulnerabilidades que se hayan dejado abiertas.

Resulta de especial interés para este trabajo el intento de restauración de la SSDT. Esta vulnerabilidad puede ser resuelta añadiendo a los servicios del sistema que son chequeados por el driver, el servicio encargado de obtener un manipulador a un objeto sección. Este servicio se conoce como *ZwOpenSection()* y en el hook instalado se debe verificar si el objeto sección que se esta intentando acceder es *device**physicalmemory.*

CAPÍTULO 4. PRUEBA DE CONCEPTO.

Introducción.

En este capítulo se desarrolla una prueba de concepto de los métodos de intercepción propuestos a lo largo de esta investigación con el objetivo de evitar las acciones malignas detectadas como sospechosas. No se pretende otorgar una versión definitiva sino solo una versión de prueba que sirva de base de conocimientos al desarrollo de una solución final como complemento de un sistema antivirus.

4.1 Características de la prueba.

Para probar los métodos propuestos durante este trabajo se desarrolló un driver nombrado *syshook.sys* y una pequeña aplicación de control S*yshookManager.exe* encargada de manejar el desempeño del driver. El método de comunicación entre el driver y la aplicación es a través de códigos de control. Un código de control no es más que un mensaje que se le envía al driver desde el modo usuario para mantener la comunicación.

El driver *syshook.sys* permite controlar la ejecución de los servicios del sistema detectados en este trabajo como posibles candidatos a ser usados por los programas malignos, posibilitando la detección e intercepción de las tres acciones malignas que son referenciadas en el primer capítulo. Los métodos de intercepción son exactamente aquellos que se plantean en las conclusiones del segundo capítulo, con adición de la seguridad necesaria para evitar las vulnerabilidades que se analizan en el tercer capítulo.

La aplicación de control S*yshookManager.exe* es una aplicación de línea de comandos que posibilita instalar el driver y desinstalarlo, y añade llaves del registro a la lista de llaves restringidas utilizadas por el driver. Esta misma aplicación es la encargada de notificar al usuario de cada acción detectada y de informarle al driver como accionar de acuerdo a la respuesta del usuario.

Para la compilación de la prueba se utilizaron los siguientes softwares, DriverStudio 3.2, el IFS-Kit 2003 y el Visual Studio.NET 2003. El ambiente para la prueba fue el sistema operativo Windows XP SP2 en una

computadora con un microprocesador de arquitectura Intel con 2.4 GHz de frecuencia y 248 MB en la memoria RAM.

4.2 SyshookManager.exe.

La aplicación de control S*yshookManager.exe* registra el driver de modo kernel *syshook.sys* y lo inicia. Envía el código de control adecuado para que el driver instale un hook en el registro y el hook para los demás servicios. Luego establece cuales son las llaves del registro a las que se les restringirá el acceso. y crea seis hilos de ejecución que permanecerán esperando por la ocurrencia de un servicio del sistema cada uno. Cuando el usuario lo desea puede detener el driver presionando la tecla “q”. El funcionamiento descrito se muestra en la figura 4.1.

El principal funcionamiento de la aplicación de control lo realizan los hilos de ejecución que se encuentran en espera del aviso de una posible acción maligna. Este aviso ocurre a través de objetos eventos del sistema creados durante la inicialización de la aplicación de control y enviados al driver a través de códigos de control. El driver activa cada objeto evento de acuerdo al servicio que haya sido interceptado.

La utilización de un objeto evento para cada servicio interceptado parte de que en caso de utilizarse un mismo evento para al menos dos servicios se corre el riesgo de que durante la intercepción del primero de ellos en el kernel y durante el tiempo que en el modo usuario se esta realizando la comunicación para decidir la acción a realizar, ocurra otra acción y la activación correspondiente se pierda. Esto puede ser resuelto mediante semáforos o mediante una complicada sincronización, pero no forma objetivo de esta prueba, sino lograr la efectividad de cada método de una forma clara y que permita medir fácilmente su efectividad.

Una vez ocurrido el evento correspondiente para cada hilo, el funcionamiento es similar para todos ellos. Se envía un código de control al driver pidiendo la información relacionada con el servicio interceptado. En el momento que retorna esta información se le muestra al usuario de una manera sencilla, y de acuerdo a la decisión de éste, se le enviará otro código de control al driver indicándole si se permite o no la acción interceptada. La figura 4.2 muestra como ocurre este proceso para el intento de acceso al registro.

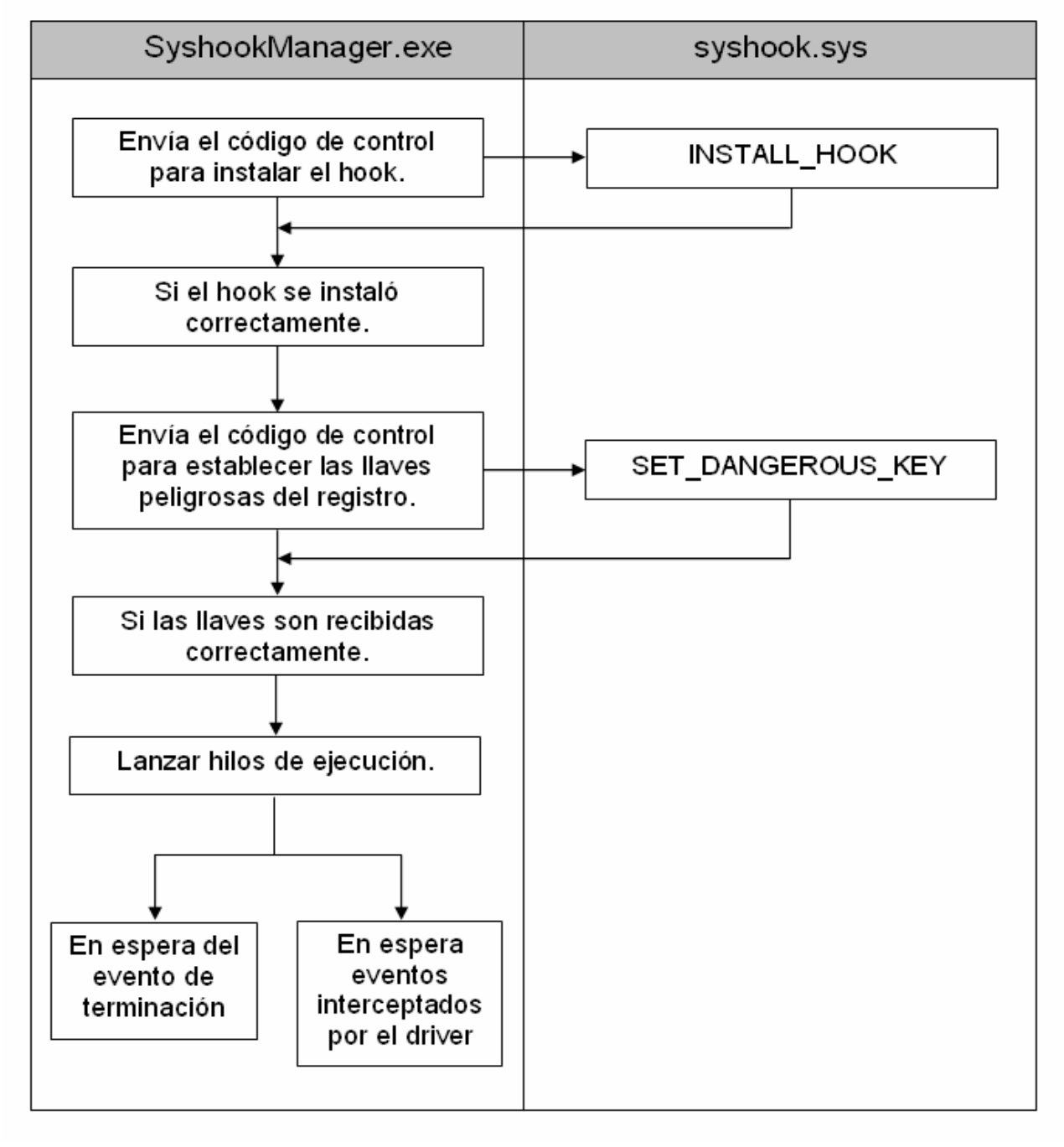

Figura 4.1. *SyshookManager.exe*. Funcionamiento general.

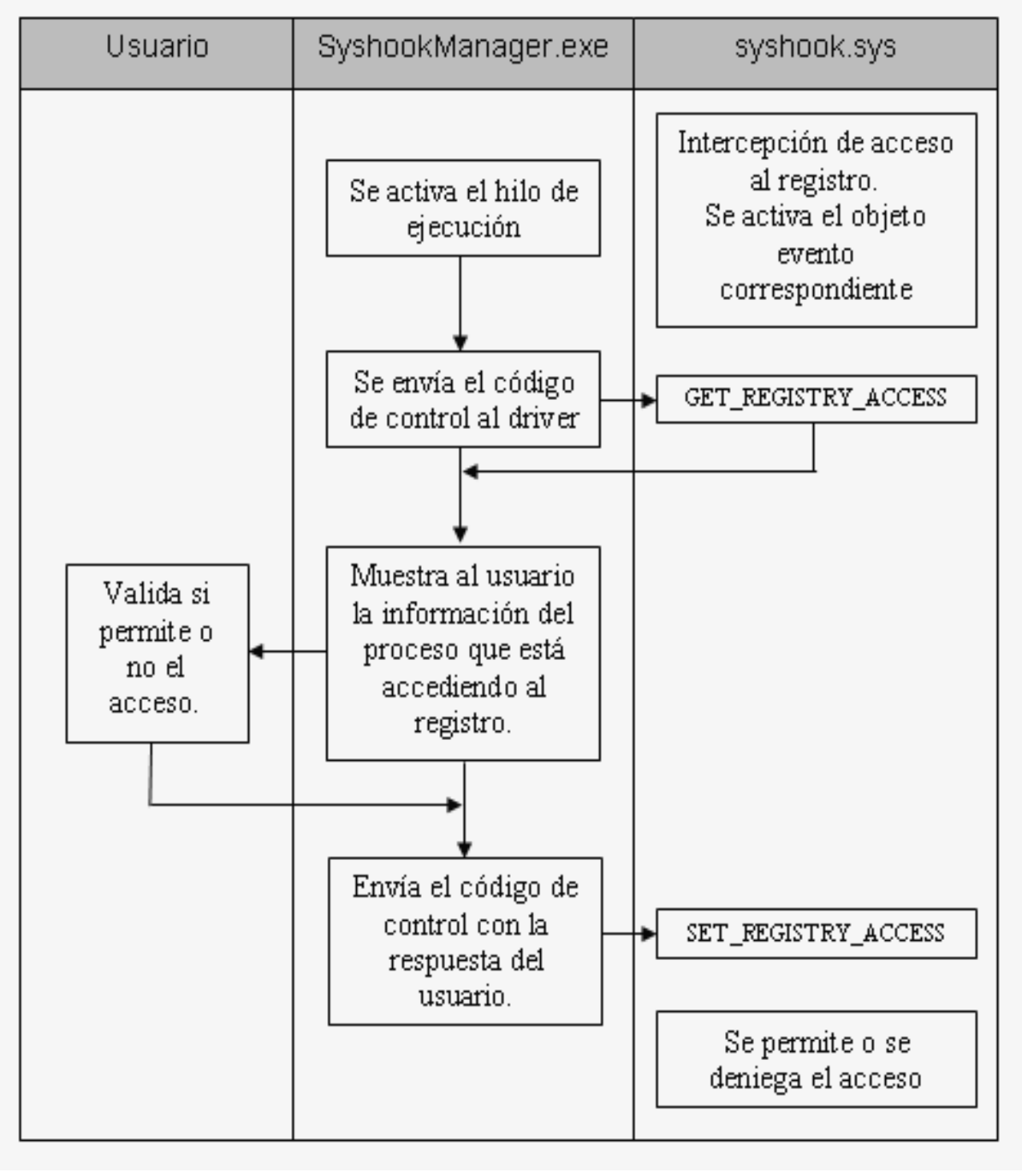

Figura 4.2. Proceso de intercepción de accesos al registro.

La figura 4.3 muestra la aplicación SyshookManager.exe en funcionamiento y la figura 4.4 muestra la información que se le muestra al usuario cuando está ocurriendo una acción maligna en este caso para el acceso al registro. Una versión mas avanzada de esta aplicación podría crear reglas de trabajo que indicarían la acción a tomar por defecto de acuerdo a decisiones anteriores.

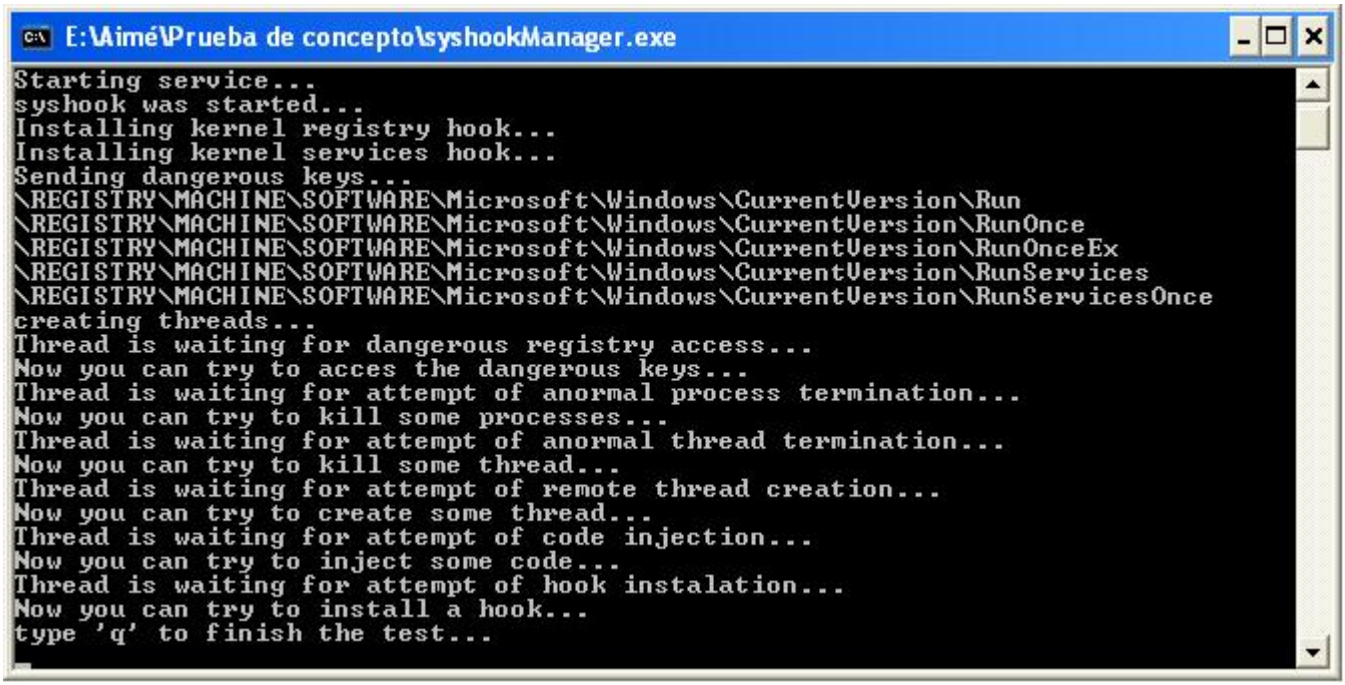

Figura 4.3. Aplicación SyshookManager.exe.

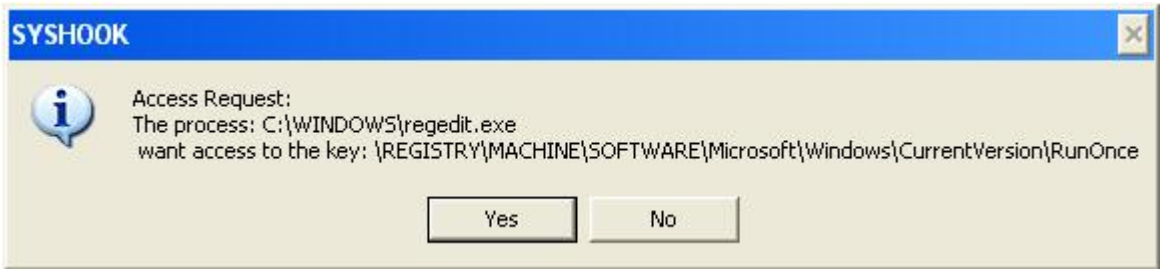

Figura 4.4. Notificación al usuario de acceso al registro.

4.3 Syshook.sys.

Un driver no es una aplicación común. Su funcionamiento no es continuo y puede correr en el espacio de direcciones lógicas de cualquier proceso en el sistema, esto quiere decir que puede estar ejecutándose en modo usuario cualquier proceso mientras el driver ejecuta una función determinada. Estas funciones no son más que respuestas a eventos ocurridos en el sistema o mensajes del sistema o del modo usuario. La forma más común de estas funciones son los códigos de control. Para dar a conocer el funcionamiento del driver desarrollado se indicarán las acciones realizadas por este en cada código de control recibido.

INSTALL_REGISTRY_HOOK: En este código de control se instala el hook del registro.

INSTALL_SERVICES_HOOK: Permite instalar el hook para los demás servicios que serán interceptados.

UNINSTALL_HOOK: Este código de control es el encargado de desinstalar los hooks si es que están instalados.

SET_REGISTRY_KEY: Mediante este código de control se establecen las llaves que constituyen llaves peligrosas para el usuario. Estas llaves son enviadas todas concatenadas desde el modo usuario en un buffer. En el driver se separan y se guardan en una lista de llaves peligrosas.

GET_REGISTRY_ACCESS: Aquí se toman los datos del acceso al registro correspondiente y se envían a la aplicación *SyshookManager.exe*.

SET_REGISTRY_ACCESS: En este código de control se valida la acción que desea llevar a cabo el usuario con el proceso que está intentando hacer cambio en alguna llave peligrosa del registro y se continúa la ejecución de acuerdo a la decisión tomada.

GET_CREATE_THREAD: Aquí se toman los datos del hilo remoto que se intenta crear y del proceso que lo intenta crear y se envían a la aplicación *SyshookManager.exe*.

SET_CREATE_THREAD: En este código de control se valida la acción que desea llevar a cabo el usuario con el proceso que está intentando crear un hilo remoto de ejecución y se continúa la ejecución de acuerdo a la decisión tomada.

GET_WRITE_VIRTUAL_MEMORY: Aquí se toman los datos del proceso que intenta escribir en la memoria virtual de otro proceso y los del proceso en el que se intenta copiar y se envían a la aplicación *SyshookManager.exe*.

SET_WRITE_VIRTUAL_MEMORY: En este código de control se valida la acción que desea llevar a cabo el usuario con el proceso que está intentando copiar en otro entorno virtual y se continúa la ejecución de acuerdo a la decisión tomada.

GET_TERMINATE_PROCESS: Aquí se toman los datos del proceso que se intenta terminar y del que está realizando la acción y se envían a la aplicación *SyshookManager.exe*.

SET_ TERMINATE_PROCESS: En este código de control se valida la acción que desea llevar a cabo el usuario con el proceso que se está intentando terminar y se continúa la ejecución de acuerdo a la decisión tomada.

GET_TERMINATE_THREAD: Aquí se toman los datos del hilo que se intenta terminar y del proceso que está llevando a cabo la acción y se envían a la aplicación *SyshookManager.exe*.

SET_ TERMINATE_THREAD: En este código de control se valida la acción que desea llevar a cabo el usuario con el hilo que se está intentando terminar y se continúa la ejecución de acuerdo a la decisión tomada.

GET_GLOBAL_HOOK: Aquí se toman los datos del proceso que está intentando instalar un hook global y se envía a la aplicación, en caso de que sea un hook en otro proceso se toman también los datos del proceso blanco sobre el cual se está llevando a cabo la acción y se envían a la aplicación *SyshookManager.exe*.

SET_ GLOBAL_HOOK: En este código de control se valida la acción que desea llevar a cabo el usuario con el proceso que está instalando el hook y se continúa la ejecución de acuerdo a la decisión tomada.

4.4 Resultados de la prueba.

Se puso en funcionamiento el sistema descrito realizándose pruebas de penetración al registro e intentándose la inyección de código y la terminación de procesos. Todas estas pruebas se realizaron tanto manualmente como automáticamente por medio de programas sencillos. Además se intentaron ataques por fuerza bruta intentando encontrar vulnerabilidades de tiempo de chequeo y tiempo de uso, así como se intentó restaurar la SSDT directamente.

Como resultados de la prueba de conceptos se tienen los siguientes:

1. Intercepción de los accesos indeseados al registro.
2. Intercepción de la inyección de código.
3. Intercepción de la terminación anormal de hilos y procesos.
4. No se detectó fallo en la ejecución del driver o la aplicación de control
5. En caso de intento de restauración de la SSDT se detectó y fue evitado.

Conclusiones.

En este capítulo se realizó una prueba de concepto que pone en práctica la investigación realizada y que sirve como punto de partida para el desarrollo de futuras aplicaciones que protejan a los sistemas informáticos mediante la defensa preventiva.

Los resultados de manera general fueron los que se esperaban, logrando los objetivos propuestos de antemano a su realización.

CONCLUSIONES.

En este trabajo se ha realizado un estudio de las acciones malignas más comunes y los métodos posibles de intercepción para cada una de estas.

Los métodos resultantes del análisis son los que se muestran en las conclusiones del capítulo dos, más los métodos de protección incorporados en el capítulo tres:

1. Utilizar la función *CmRegisterCallback()* para registrar una función hook que intercepte las acciones de escritura en llaves peligrosas del registro. Para ello la función hook debe obtener el nombre de la llave a la que se está accediendo y verificar si es peligrosa o no. El driver entonces tomará la acción que se le haya predefinido o preguntará al usuario la acción a tomar. Todo esto es a consideración del diseñador del software de protección.

2. Realizar un hook en la SSDT para los servicios del sistema que se muestran a continuación:

   - *ZwCreateThread()* (ntoskrnl.exe).
   - *ZwWriteVirtualMemory()* (ntoskrnl.exe).
   - *ZwTerminateProcess()* (ntoskrnl.exe).
   - *ZwTerminateThread()* (ntoskrnl.exe).

   Para estos servicios se debe obtener en cada caso el proceso que realiza la acción y el proceso objeto sobre el que se ejecutará. El driver entonces debe tomar la acción que haya sido seleccionada por defecto o preguntar al usuario la acción a tomar. Todo esto es a consideración del diseñador del software de protección.

3. Realizar un hook en la SSDTS para el servicio del sistema que se muestran a continuación:

   - NtUserSetWindowsHookEx() (win32k.sys)

Para este servicio solo será necesario detectar si el hook que desea instalar esta función en el sistema es de alcance global o local, siendo el primero el centro de atención en cuanto a seguridad.

4. Realizar un hook adicional en la SSDT en el servicio del sistema *ZwOpenSection()* a fin de evitar la restauración de la SSDT y por lo tanto la eliminación de la protección.

A partir de los resultados obtenidos se procederá a añadir al producto antivirus cubano la protección de accesos al registro, mientras que las otras protecciones permanecen aún en estudio. Los códigos desarrollados para las pruebas de concepto servirán como base para el desarrollo del producto final en el antivirus.

Tanto los servicios detectados por ser frecuentemente utilizados por los programas malignos, como los métodos de intercepción de estos deben ser revisados en cada nueva versión del sistema operativo, con el objetivo de evitar futuras incompatibilidades.

BIBLIOGRAFÍA

Explicación del Registro de Windows. [en línea].
http://www.varelaenred.com.ar/explicacion%20registro%20de%20win.htm [Consulta: Enero 18, 2007].

Introducción al registro de Windows. [en línea].
http://www.svetlian.com/windows/temas_registro.htm [Consulta: Enero 20, 2007].

Registry Callbacks. [en línea]. 2001.
http://www.i.u-tokyo.ac.jp/edu/training/ss/lecture/new-documents/Projects/RegistryCallbacksSpec.pdf
[Consulta: Enero 22, 2007].

Systems Internals Tips and Trivia. [en línea]. November, 2006.
http://www.microsoft.com/technet/sysinternals/information/tipsandtrivia.mspx [Consulta: Febrero 5, 2007].

Virus Informáticos [Clasificaciones]. [en línea]. Enero, 2004.
http://www.network-press.org/?virus_informaticos_concepto [Consulta: Septiembre 24, 2006].

Bassov, A. *Hooking the kernel directly*. [en línea]. April, 2006.
http://www.codeproject.com/system/soviet_direct_hooking.asp [Consulta: Noviembre 11, 2007].

Bishop, M. *Checking for Race Conditions in File Accesses.* [en línea]. February, 2003.
http://seclab.cs.ucdavis.edu/projects/vulnerabilities/scriv/1996-compsys.pdf [Consulta: Febrero 6, 2007].

Dabak, P. *Hooking Windows NT System Services.* [en línea]. October, 1999.
http://www.windowsitlibrary.com/Content/356/06/2.html [Consulta: Noviembre 5, 2007]

Dabak, P. *Writing Windows NT Device Drivers*. [en línea]. October, 1999.
http://www.windowsitlibrary.com/Content/356/02/toc.html [Consulta: Noviembre 9, 2007].

Dabak, P. *Undocumented Windows NT.* [en línea]. October, 1999. http://www.windowsitlibrary.com/Documents/Book.cfm?DocumentID=356 [Consulta: Noviembre 1, 2007].

Ivanov, I. *Detecting Windows NT/2K process execution*. [en línea]. March, 2002. http://www.codeproject.com/threads/procmon.asp [Consulta: Noviembre 2, 2007].

Keong, T. C. *Defeating Kernel Native API Hookers by Direct KiServiceTable Restoration.* [en línea]. July, 2004. http://www.packetstormsecurity.org/hitb04/hitb04-chew-keong-tan.pdf [Consulta: Febrero 11, 2007].

Keong, T. C. *DiamondCS Process Guard Can Be Disabled by Direct Service Table Restoration.* [en línea]. July, 2004. http://www.security.org.sg/vuln/procguard.html [Consulta: Febrero 6, 2007].

Opferman, T. *Driver Development Part 1: Introduction to Drivers.* [en línea]. February, 2005. http://www.codeproject.com/system/driverdev.asp [Consulta: Noviembre 10, 2006].

Russinovich, M. *RegMon for Windows v7.04.* [en línea]. November, 2006. http://www.microsoft.com/technet/sysinternals/utilities/Regmon.mspx [Consulta: Enero 23, 2007]

Skywing. *What Were They Thinking? Anti-Virus Software Gone Wrong.* [en línea]. May, 2006. http://www.uninformed.org/?v=4&a=4.Z [Consulta: Febrero 15, 2007].

GLOSARIO

Antivirus: Los antivirus son programas cuya función es detectar y eliminar los programas malignos.

Depurador: En inglés debug. Es un programa que permite la edición y creación de otros programas escritos en algún lenguaje. Así como posibilita al que lo usa una vía para encontrar los errores que contenga el programa desarrollado.

Descriptor de Compuerta de Interrupción (IGD): La IDT está formada por 256 vectores que constituyen cada uno el descriptor de la interrupción indicada.

International Computer Security Association (ICSA): formalmente llamada NCSA (National Computer Security Association), esta organización está consagrada a los asuntos de seguridad en las computadoras de las corporaciones, asociaciones y las agencies de gobierno del mundo. Está dedicada a, continuamente, mejorar la seguridad por medio de certificaciones, intercambio de conocimientos y diseminación de la información. La organización fue fundada en 1989 y se encuentra en Carlisle, Pennsylvania.

Interrupción: Es un mensaje que se envía cuando se tiene una información que necesita el procesador mediante el cual se interrumpe el procesamiento corriente y se salta a ejecutar una parte de código determinada con anterioridad. Cuando un procesador está esperando recibir información de un cierto dispositivo externo, tiene dos formas básicas de hacerlo: mirando continuamente si hay información disponible (modo de espera o modo "polling"), o dejando la posibilidad de que sea el dispositivo el que avise cuando la tenga preparada (modo interrupción).

Kernel: Es la parte fundamental de un sistema operativo o el núcleo del mismo. Es el software responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma más básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema. Acceder al hardware directamente puede ser realmente complejo, por lo que el kernel permite implementar una serie de abstracciones del hardware. Esto permite esconder la complejidad, y

proporciona una interfaz limpia y uniforme al hardware subyacente, lo que facilita su uso para el programador.

Servicios del sistema: Son funciones que brindan la posibilidad de realizar una acción determinada en el sistema operativo. Sin importar cuan sencilla o compleja pueda ser la tarea a realizar, siempre se traducirá a una llamada al servicio del sistema correspondiente para realizar la misma.

Sistema operativo: Es un conjunto de programas que facilitan el trabajo del usuario permitiéndole una comunicación amena con la computadora. Es la capa de software que posibilita el manejo eficiente de la computadora.

Tabla de Descriptores de Interrupción (IDT): Es una estructura de datos usada por la arquitectura x86 de Intel y AMD64 para implementar la tabla de vectores de interrupción. La IDT es usada por el procesador para conocer la respuesta correspondiente a una interrupción o una excepción. La IDT consiste en un arreglo de 256 vectores.